Одиннадцатая независимая научно-практическая конференция «Разработка ПО 2015»

22 - 24 октября, Москва

Повседневный C++

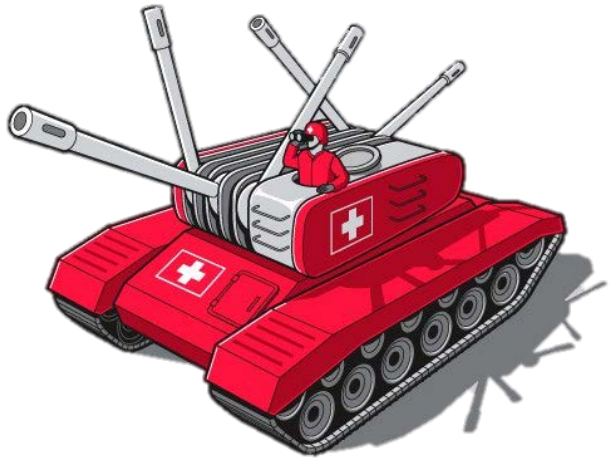
Михаил Матросов

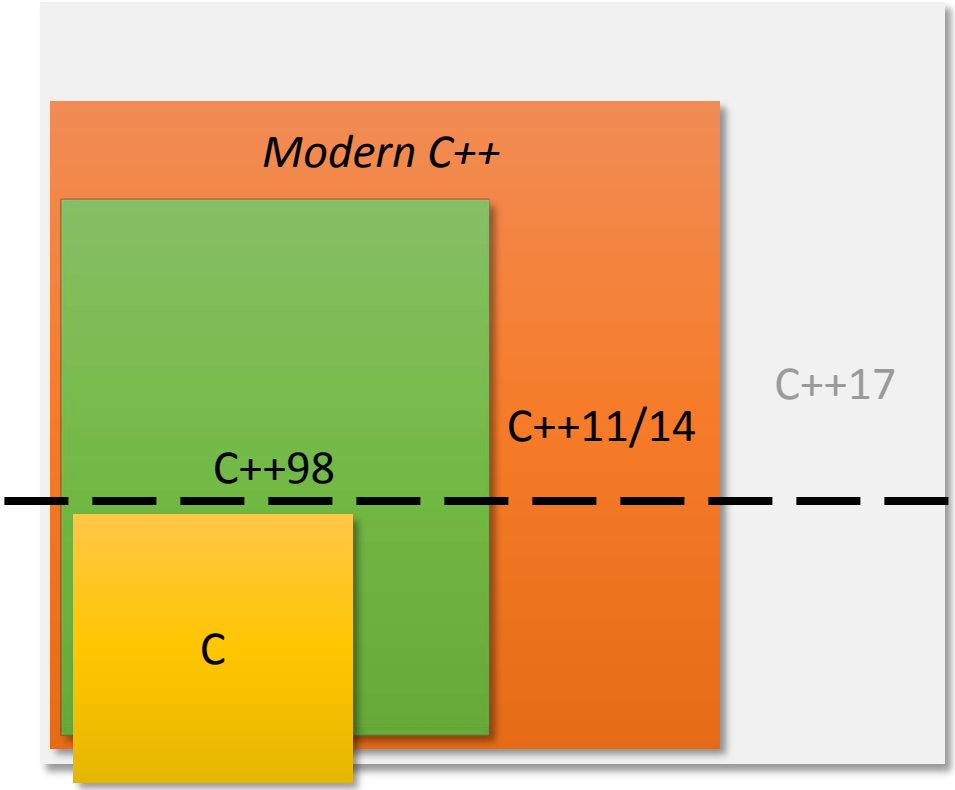
Align Technology

mmatrosov@aligntech.com

mikhail.matrosov@gmail.com


WENGER





High level

Expert level

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

Bjarne Stroustrup

Классический слайд с телом и заголовком

High level:

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

Expert level:

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

```
bool doCompare(const Bakery* oldBakery, const Bakery* newBakery)
{
  if (!oldBakery || !newBakery)
  {
    return false;
  }

  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;
  std::vector<std::pair<Cookie, int>> diff;

  collectCookies(*oldBakery, oldCookies);
  collectCookies(*newBakery, newCookies);

  std::sort(oldCookies.begin(), oldCookies.end(), compareCookies);
  std::sort(newCookies.begin(), newCookies.end(), compareCookies);

  auto oldIt = oldCookies.begin();
  auto newIt = newCookies.begin();

  while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
```

```cpp
bool doCompare(const Bakery* oldBakery, const Bakery* newBakery)
{
  if (!oldBakery || !newBakery)
  {
    return false;
  }

  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;
  std::vector<std::pair<Cookie, int>> diff;

  collectCookies(*oldBakery, oldCookies);
  collectCookies(*newBakery, newCookies);

  std::sort(oldCookies.begin(), oldCookies.end(), compareCookies);
  std::sort(newCookies.begin(), newCookies.end(), compareCookies);

  auto oldIt = oldCookies.begin();
  auto newIt = newCookies.begin();

  while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
  {
    if (compareCookies(*oldIt, *newIt))
    {
      diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
      ++oldIt;
    }
    else if (compareCookies(*newIt, *oldIt))
    {
      diff.push_back(std::pair<Cookie, int>(*newIt, 2));
      ++newIt;
    }
    else
    {
      ++oldIt;
      ++newIt;
    }
  }

  if (oldIt != oldCookies.end())
  {
    for (; oldIt < oldCookies.end(); oldIt++)
    {
      diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
    }
  }

  if (newIt != newCookies.end())
  {
    for (; newIt < newCookies.end(); newIt++)
    {
      diff.push_back(std::pair<Cookie, int>(*newIt, 2));
    }
  }

  for (const auto& item : diff)
  {
    std::cout << "Bakery #" << item.second <<
      " has a different cookie \"" << item.first.name <<
      "\" (weight=" << item.first.weight <<
      ", volume=" << item.first.volume <<
      ", tastiness=" << item.first.tastiness << ")." << std::endl;
  }

  if (diff.size() > 0)
    return false;
  return true;
}
```

```cpp
bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  auto oldCookies = collectCookies(oldBakery);
  auto newCookies = collectCookies(newBakery);

  auto reportLost = [](const Cookie& cookie) {
    std::cout << "We've lost a friend: " << cookie << "." << std::endl;
  };
  auto reportAppeared = [](const Cookie& cookie) {
    std::cout << "We got a new friend: " << cookie << "." << std::endl;
  };

  boost::range::set_difference(oldCookies, newCookies,
                               boost::make_function_output_iterator(reportLost));
  boost::range::set_difference(newCookies, oldCookies,
                               boost::make_function_output_iterator(reportAppeared));

  return oldCookies == newCookies;
}
```

"Повседневный C++", Михаил Матросов, CEE-SECR 2015

```
bool doCompare(const Bakery* oldBakery, const Bakery* newBakery)
{
  if (!oldBakery || !newBakery)
  {
    return false;
  }

  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;
  std::vector<std::pair<Cookie, int>> diff;

  collectCookies(*oldBakery, oldCookies);
  collectCookies(*newBakery, newCookies);

  // ...
```

```cpp
bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;
  std::vector<std::pair<Cookie, int>> diff;

  collectCookies(oldBakery, oldCookies);
  collectCookies(newBakery, newCookies);

  // ...
```


2.2m

```
// ...
std::sort(oldCookies.begin(), oldCookies.end(), compareCookies);
std::sort(newCookies.begin(), newCookies.end(), compareCookies);

auto oldIt = oldCookies.begin();
auto newIt = newCookies.begin();

while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
{
  if (compareCookies(*oldIt, *newIt))
  {
    diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
    ++oldIt;
  }
  else if (compareCookies(*newIt, *oldIt))
  {
    // ...
```

```cpp
struct Cookie
{
  double weight;
  double volume;
  std::string name;
  int tastiness;
};

bool compareCookies(const Cookie& a, const Cookie& b)
{
  if (a.tastiness != b.tastiness)
    return a.tastiness < b.tastiness;
  if (a.weight != b.weight)
    return a.weight < b.weight;
  if (a.volume != b.volume)
    return a.volume < b.volume;
  return a.name < b.name;
}
```

```cpp
struct Cookie
{
  double weight;
  double volume;
  std::string name;
  int tastiness;
};

bool operator<(const Cookie& a, const Cookie& b)
{
  if (a.tastiness != b.tastiness)
    return a.tastiness < b.tastiness;
  if (a.weight != b.weight)
    return a.weight < b.weight;
  if (a.volume != b.volume)
    return a.volume < b.volume;
  return a.name < b.name;
}
```

```
// ...
std::sort(oldCookies.begin(), oldCookies.end(), compareCookies);
std::sort(newCookies.begin(), newCookies.end(), compareCookies);

auto oldIt = oldCookies.begin();
auto newIt = newCookies.begin();

while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
{
  if (compareCookies(*oldIt, *newIt))
  {
    diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
    ++oldIt;
  }
  else if (compareCookies(*newIt, *oldIt))
  {
    // ...
```

```
// ...
std::sort(oldCookies.begin(), oldCookies.end());
std::sort(newCookies.begin(), newCookies.end());

auto oldIt = oldCookies.begin();
auto newIt = newCookies.begin();

while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
{
  if (*oldIt < *newIt)
  {
    diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
    ++oldIt;
  }
  else if (*newIt < *oldIt)
  {
    // ...
```

```cpp
bool operator<(const Cookie& a, const Cookie& b)
{
  if (a.tastiness != b.tastiness)
    return a.tastiness < b.tastiness;
  if (a.weight != b.weight)
    return a.weight < b.weight;
  if (a.volume != b.volume)
    return a.volume < b.volume;
  return a.name < b.name;
}
```

```cpp
bool operator<(const Cookie& a, const Cookie& b)
{
  return std::make_tuple(a.tastiness, a.weight, a.volume, a.name) <
         std::make_tuple(b.tastiness, b.weight, b.volume, b.name);
}
```

```cpp
struct Cookie
{
  double weight;
  double volume;
  std::string name;
  int tastiness;
};

bool operator<(const Cookie& a, const Cookie& b)
{
  return std::make_tuple(a.tastiness, a.weight, a.volume, a.name) <
         std::make_tuple(b.tastiness, b.weight, b.volume, b.name);
}
```

```cpp
struct Cookie
{
  double weight;
  double volume;
  std::string name;
  int tastiness;
};

bool operator<(const Cookie& a, const Cookie& b)
{
  return std::tie(a.tastiness, a.weight, a.volume, a.name) <
         std::tie(b.tastiness, b.weight, b.volume, b.name);
}
```

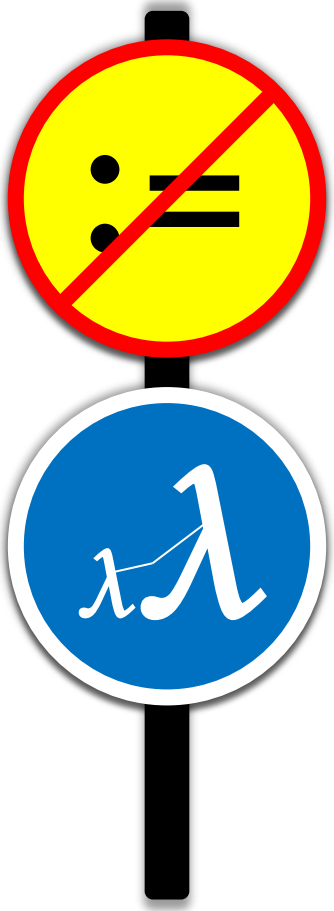
```cpp
struct Cookie
{
  double weight;
  double volume;
  std::string name;
  int tastiness;

  auto rank() const
  {
    return std::tie(tastiness, weight, volume, name);
  }
  bool operator<(const Cookie& rhs) const
  {
    return rank() < rhs.rank();
  }
};
```

```cpp
struct Cookie
{
  double weight;
  double volume;
  std::string name;
  int tastiness;

  auto rank() const  // -> std::tuple<double&, double&, std::string&, int&>
  {
    return std::tie(tastiness, weight, volume, name);
  }
  bool operator<(const Cookie& rhs) const
  {
    return rank() < rhs.rank();
  }
};
```

```
bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;

  collectCookies(oldBakery, oldCookies);
  collectCookies(newBakery, newCookies);

  // ...
```

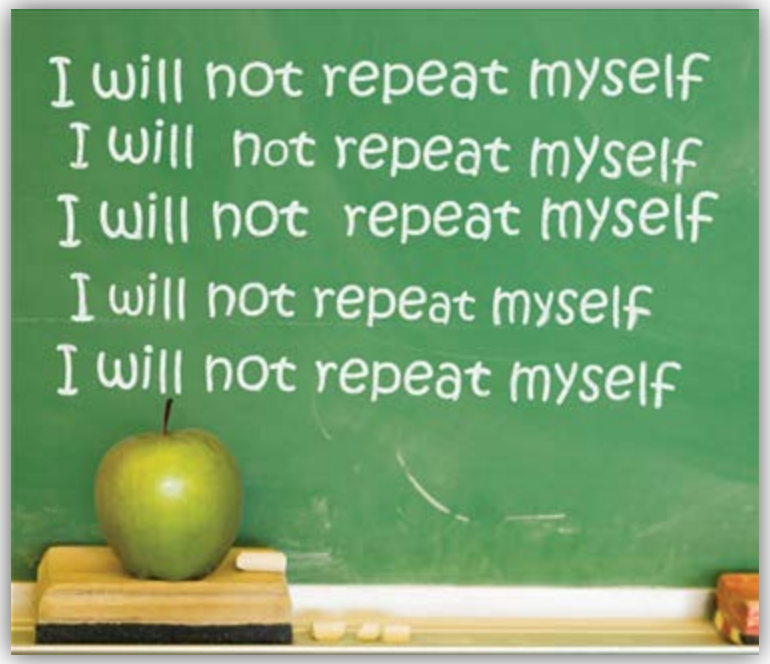
```
void collectCookies(const Bakery& bakery, std::vector<Cookie>& o_cookies);

bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;

  collectCookies(oldBakery, oldCookies);
  collectCookies(newBakery, newCookies);

  // ...
```

```cpp
std::vector<Cookie> collectCookies(const Bakery& bakery);

bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies = collectCookies(oldBakery);
  std::vector<Cookie> newCookies = collectCookies(newBakery);

  // ...
```



```
std::vector<Cookie> collectCookies(const Bakery& bakery)
{
  std::vector<Cookie> cookies;
  // ...
  // ...
  return cookies;
}

bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies = collectCookies(oldBakery);
  std::vector<Cookie> newCookies = collectCookies(newBakery);

  std::sort(oldCookies.begin(), oldCookies.end());
  std::sort(newCookies.begin(), newCookies.end());

  // ...
```

```cpp
std::vector<Cookie> collectCookies(const Bakery& bakery)
{
  std::vector<Cookie> cookies;
  // ...
  std::sort(cookies.begin(), cookies.end());
  return cookies;
}

bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies = collectCookies(oldBakery);
  std::vector<Cookie> newCookies = collectCookies(newBakery);

  // ...
```


I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself

```cpp
std::vector<std::pair<Cookie, int>> diff;

// ...

if (*oldIt < *newIt)
{
  diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
  ++oldIt;
}
else if (*newIt < *oldIt)
{
  diff.push_back(std::pair<Cookie, int>(*newIt, 2));
  ++newIt;
}
```

```cpp
std::vector<std::pair<Cookie, int>> diff;

// ...

if (*oldIt < *newIt)
{
  diff.push_back(std::make_pair(*oldIt, 1));
  ++oldIt;
}
else if (*newIt < *oldIt)
{
  diff.push_back(std::make_pair(*newIt, 2));
  ++newIt;
}
```

```cpp
std::vector<std::pair<Cookie, int>> diff;

// ...

if (*oldIt < *newIt)
{
  diff.emplace_back(*oldIt, 1);
  ++oldIt;
}
else if (*newIt < *oldIt)
{
  diff.emplace_back(*newIt, 2);
  ++newIt;
}
```

```
for (const auto& item : diff)  // item is std::pair<Cookie, int>
{
  std::cout << "Bakery #" << item.second <<
    " has a different cookie \"" << item.first.name <<
    "\" (weight=" << item.first.weight <<
    ", volume=" << item.first.volume <<
    ", tastiness=" << item.first.tastiness << ")." << std::endl;
}
```

```
std::ostream& operator<<(std::ostream& stream, const Cookie& cookie)
{
  return stream << "\"" << cookie.name <<
    "\" (weight=" << cookie.weight <<
    ", volume=" << cookie.volume <<
    ", tastiness=" << cookie.tastiness << ")";
}
```

```
for (const auto& item : diff)  // item is std::pair<Cookie, int>
{
  std::cout << "Bakery #" << item.second <<
    " has a different cookie " << item.first << "." << std::endl;
}
```

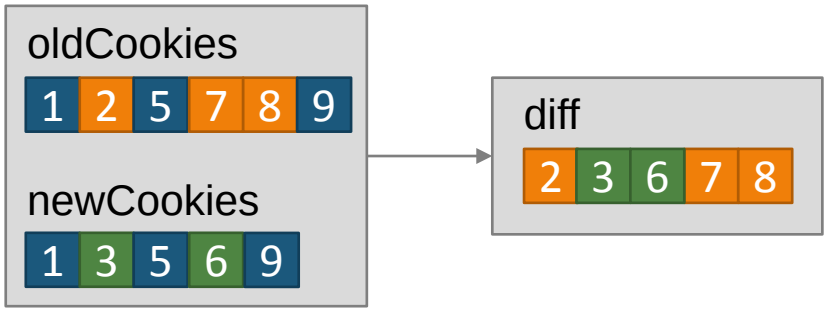
```
if (diff.size() > 0)
  return false;
return true;
```

```
return diff.empty();
```


Evan
Williams
SINCE 1783
ORIGINAL SOUTHERN
EGG
NOG

```cpp
std::vector<std::pair<Cookie, int>> diff;

auto oldIt = oldCookies.begin();
auto newIt = newCookies.begin();

while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
{
  if (*oldIt < *newIt)
  {
    diff.emplace_back(*oldIt, 1);
    ++oldIt;
  }
  else if (*newIt < *oldIt)
  {
    diff.emplace_back(*newIt, 2);
    ++newIt;
  }
  else
  {
    ++oldIt;
    ++newIt;
  }
}

for (; oldIt != oldCookies.end(); oldIt++)
{
  diff.emplace_back(*oldIt, 1);
}

for (; newIt != newCookies.end(); newIt++)
{
  diff.emplace_back(*newIt, 2);
}
```

Ошибка. Должно быть &&



```
std::vector<Cookie> lostCookies;
std::set_difference(oldCookies.begin(), oldCookies.end(),
                    newCookies.begin(), newCookies.end(),
                    std::back_inserter(lostCookies));

std::vector<Cookie> appearedCookies;
std::set_difference(newCookies.begin(), newCookies.end(),
                    oldCookies.begin(), oldCookies.end(),
                    std::back_inserter(appearedCookies));
```

```
std::vector<Cookie> lostCookies;
boost::range::set_difference(oldCookies, newCookies,
                             std::back_inserter(lostCookies));

std::vector<Cookie> appearedCookies;
boost::range::set_difference(newCookies, oldCookies,
                             std::back_inserter(appearedCookies));
```

```
std::vector<Cookie> lostCookies;
boost::range::set_difference(oldCookies, newCookies,
                             std::back_inserter(lostCookies));

std::vector<Cookie> appearedCookies;
boost::range::set_difference(newCookies, oldCookies,
                             std::back_inserter(appearedCookies));

for (const auto& cookie : lostCookies)
{
  std::cout << "We've lost a friend: " << cookie << "." << std::endl;
}
for (const auto& cookie : appearedCookies)
{
  std::cout << "We got a new friend: " << cookie << "." << std::endl;
}
```

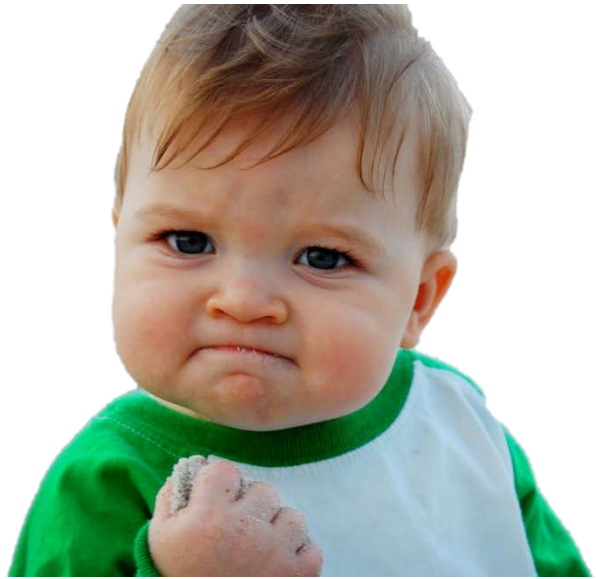
```
auto reportLost = [](const Cookie& cookie) {
  std::cout << "We've lost a friend: " << cookie << "." << std::endl;
};
auto reportAppeared = [](const Cookie& cookie) {
  std::cout << "We got a new friend: " << cookie << "." << std::endl;
};

boost::range::set_difference(oldCookies, newCookies,
                             boost::make_function_output_iterator(reportLost));
boost::range::set_difference(newCookies, oldCookies,
                             boost::make_function_output_iterator(reportAppeared));
```

```cpp
bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  std::vector<Cookie> oldCookies = collectCookies(oldBakery);
  std::vector<Cookie> newCookies = collectCookies(newBakery);

  auto reportLost = [](const Cookie& cookie) {
    std::cout << "We've lost a friend: " << cookie << "." << std::endl;
  };
  auto reportAppeared = [](const Cookie& cookie) {
    std::cout << "We got a new friend: " << cookie << "." << std::endl;
  };

  boost::range::set_difference(oldCookies, newCookies,
                               boost::make_function_output_iterator(reportLost));
  boost::range::set_difference(newCookies, oldCookies,
                               boost::make_function_output_iterator(reportAppeared));

  return oldCookies == newCookies;
}
```

```cpp
bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  auto oldCookies = collectCookies(oldBakery);
  auto newCookies = collectCookies(newBakery);

  auto reportLost = [](const Cookie& cookie) {
    std::cout << "We've lost a friend: " << cookie << "." << std::endl;
  };
  auto reportAppeared = [](const Cookie& cookie) {
    std::cout << "We got a new friend: " << cookie << "." << std::endl;
  };

  boost::range::set_difference(oldCookies, newCookies,
                               boost::make_function_output_iterator(reportLost));
  boost::range::set_difference(newCookies, oldCookies,
                               boost::make_function_output_iterator(reportAppeared));

  return oldCookies == newCookies;
}
```

```cpp
bool doCompare(const Bakery* oldBakery, const Bakery* newBakery)
{
  if (!oldBakery || !newBakery)
  {
    return false;
  }

  std::vector<Cookie> oldCookies;
  std::vector<Cookie> newCookies;
  std::vector<std::pair<Cookie, int>> diff;

  collectCookies(*oldBakery, oldCookies);
  collectCookies(*newBakery, newCookies);

  std::sort(oldCookies.begin(), oldCookies.end(), compareCookies);
  std::sort(newCookies.begin(), newCookies.end(), compareCookies);

  auto oldIt = oldCookies.begin();
  auto newIt = newCookies.begin();

  while ((oldIt != oldCookies.end()) || (newIt != newCookies.end()))
  {
    if (compareCookies(*oldIt, *newIt))
    {
      diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
      ++oldIt;
    }
    else if (compareCookies(*newIt, *oldIt))
    {
      diff.push_back(std::pair<Cookie, int>(*newIt, 2));
      ++newIt;
    }
    else
    {
      ++oldIt;
      ++newIt;
    }
  }

  if (oldIt != oldCookies.end())
  {
    for (; oldIt < oldCookies.end(); oldIt++)
    {
      diff.push_back(std::pair<Cookie, int>(*oldIt, 1));
    }
  }

  if (newIt != newCookies.end())
  {
    for (; newIt < newCookies.end(); newIt++)
    {
      diff.push_back(std::pair<Cookie, int>(*newIt, 2));
    }
  }

  for (const auto& item : diff)
  {
    std::cout << "Bakery #" << item.second <<
      " has a different cookie \"" << item.first.name <<
      "\" (weight=" << item.first.weight <<
      ", volume=" << item.first.volume <<
      ", tastiness=" << item.first.tastiness << ")." << std::endl;
  }

  if (diff.size() > 0)
    return false;
  return true;
}
```

```cpp
bool doCompare(const Bakery& oldBakery, const Bakery& newBakery)
{
  auto oldCookies = collectCookies(oldBakery);
  auto newCookies = collectCookies(newBakery);

  auto reportLost = [](const Cookie& cookie) {
    std::cout << "We've lost a friend: " << cookie << "." << std::endl;
  };
  auto reportAppeared = [](const Cookie& cookie) {
    std::cout << "We got a new friend: " << cookie << "." << std::endl;
  };

  boost::range::set_difference(oldCookies, newCookies,
                               boost::make_function_output_iterator(reportLost));
  boost::range::set_difference(newCookies, oldCookies,
                               boost::make_function_output_iterator(reportAppeared));

  return oldCookies == newCookies;
}
```



"Повседневный C++", Михаил Матросов, CEE-SECR 2015

Код, использующий стандартные алгоритмы и контейнеры:

- Короче
- Проще
- Надёжнее
- Обычно эффективнее
- Проще переиспользовать
- Не привязан к конкретной платформе

Советы:

- Мыслите на высоком уровне
- Код должен ясно выражать намерение
- Знайте свои инструменты и используйте их к месту

```
// references
// auto variables
// auto function return type
// move semantics
// lambda functions
operator<();
std::make_tuple();
std::tie();
```

```
std::make_pair();
std::vector<T>::emplace_back();
std::vector<T>::empty();
operator<<();
std::back_inserter();
std::set_difference();
boost::range::set_difference();
boost::make_function_output_iterator();
```

Спасибо за внимание!

- Мыслите на высоком уровне
- Код должен ясно выражать намерение
- Знайте свои инструменты и используйте их к месту

```
// references
// auto variables
// auto function return type
// move semantics
// lambda functions
operator<();
std::make_tuple();
std::tie();
```

```
std::make_pair();
std::vector<T>::emplace_back();
std::vector<T>::empty();
operator<<();
std::back_inserter();
std::set_difference();
boost::range::set_difference();
boost::make_function_output_iterator();
```