
Яндекс

Яндекс

C++ Contracts

Antony Polukhin
Полухин Антон

Author of Boost's TypeIndex, DLL, Stacktrace
Maintainer of Boost Any, Conversion, LexicalCast, Variant
ISO WG21 national body

What are we going to talk about?

What are we going to talk about?

- What the hell is Contracts?

What are we going to talk about?

- What the hell is Contracts?
- <algorithm> header

What are we going to talk about?

- What the hell is Contracts?
- <algorithm> header
- Big “O”

What are we going to talk about?

- What the hell is Contracts?
- <algorithm> header
- Big “O”
- Contracts and algorithms

Contracts

Contracts

Contracts – in development feature of the C++ language that allows to specify the function domain.

Contracts — asserts on steroids

```
void push(int x, queue & q);
```

Contracts — asserts on steroids

```
void push(int x, queue & q)
  [[expects: !q.full()]]
  [[ensures: !q.empty()]]
{
  //...
  [[assert: q.is_valid()]];
  //...
}
```

Contracts — asserts on steroids

```
void push(int x, queue & q) [[expects: !q.full()]] [[ensures: !q.empty()]] ;

queue q;

// ...
```

Contracts — asserts on steroids

```
void push(int x, queue & q) [[expects: !q.full()]] [[ensures: !q.empty()]] ;

queue q;

// ...

if (!q.full()) {

    push(10, q); // No need to check

    if (q.empty()) { // Always false

    }

} else {

    push(11, q); // Suspicious!

}
```

Contracts — asserts on steroids

```
void(const std::contract_violation &);


namespace std {
  class contract_violation {
  public:
    int line_number() const noexcept;
    const char * file_name() const noexcept;
    const char * function_name() const noexcept;
    const char * comment() const noexcept;
  };
}
```

<algorithm>

std::sort(beg, end)

Sorts in ascending order:

4	6	9	1	2	5	3	8	7	0	@

std::sort(beg, end)

Sorts in ascending order:

4	6	9	1	2	5	3	8	7	0	@

0	1	2	3	4	5	6	7	8	9	@

std::sort(beg, end)

```
template <class RandomAccessIterator>
void sort(RandomAccessIterator beg, RandomAccessIterator end)
    [[expects: beg <= end]]
    [[ensures: is_sorted(beg, end)]]
;
```

Big «O»

Big «O»

Big “O”– used to classify algorithms according to how their running time or space requirements grow as the input size N grows.

Big «O»

Big “O”– used to classify algorithms according to how their running time or space requirements grow as the input size N grows.

for (size_t i = 0; i < N; ++i) => O(N)

Big «O»

Big “O”– used to classify algorithms according to how their running time or space requirements grow as the input size N grows.

```
for (size_t i = 0; i < N; ++i)                  => O(N)
```

```
for (size_t i = 0; i < N; ++i)
        for (size_t j = 0; j < N; ++j)          => O(N²)
```

Big «O»

N	N*log(N)	N*N
2	2	4
4	8	16
8	24	64
16	64	256
32	160	1,024
64	384	4,096
128	896	16,384
256	2,048	65,536
512	4,608	262,144
1,024	10,240	1,048,576

Big «O»

- std::sort => $O(N*\log_2(N))$

Big «O»

- std::sort => $O(N*\log_2(N))$
- std::stable_sort => $O(N*\log_2^2(N))$

Big «O»

- ~~std::sort~~ => O(N*$\log_2$(N))
- ~~std::stable_sort~~ => O(N*$\log_2^2$(N))

- std::minmax_element => O(N)
- std::nth_element => ~ O(N)
- std::partial_sort => O(N*$\log_2$(S))

std::nth_element(beg, mid, end)

- Reorders the range [beg, end) so that:
  - Value pointed to by mid does not change if we sort the range [beg, end)
  - Left from mid are values less or equal to the value pointed to by mid
  - Right from mid are values grater or equal to the value pointed to

4	0	3	1	2	5	9	8	7	6	@

std::nth_element(beg, mid, end)

- Reorders the range [beg, end) so that:
  - Value pointed to by mid does not change if we sort the range [beg, end)
  - Left from mid are values less or equal to the value pointed to by mid
  - Right from mid are values grater or equal to the value pointed to

4	0	3	1	2	5	9	8	7	6	@

0	1	2	3	4	5	6	7	8	9	@

std::nth_element

Find 5 people with minimal balance

```
std::nth_element(v.begin(), v.begin() + 4, v.end());
```

Find 5 people with maximal balance

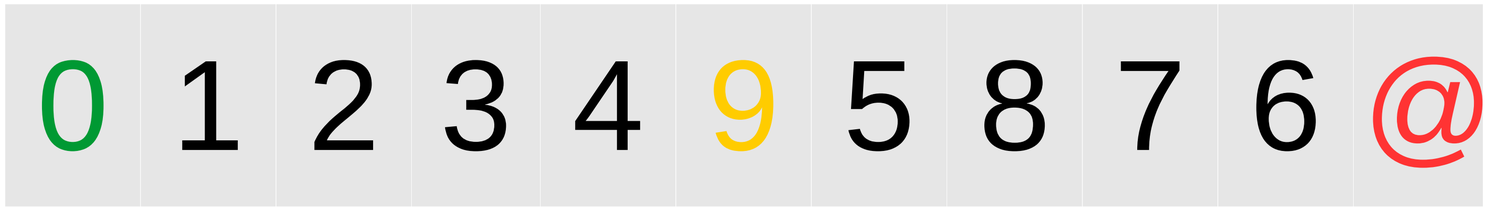
```
std::nth_element(v.begin(), v.begin() + 4, v.end(),
std::greater<>{});
```

Find the lucky 1001 caller

```
std::nth_element(v.begin(), v.begin() + 1000, v.end());
```

std::partial_sort(beg, mid, end)

- Reorders the range [beg, end) so that:
  - Range [beg, mid) won't change if we sort the whole [beg, end) range
  - Range [beg, mid) is sorted



std::partial_sort

Distribute 5 prizes according to minimal penalty points

```
std::partial_sort(v.begin(), v.begin() + 5, v.end());
```

Punish 5 latecomer scholars

```
std::partial_sort(v.begin(), v.begin() + 5, v.end(),
std::greater<>{});
```

std::minmax_element

Find richest and poorest client:

```
auto mm = std::minmax_element(v.begin(), v.end());
std::cout << *mm.first << ' ' << *mm.second << '\n';
```

The Question:

How to get a sorted list of 10 people with balance close to median?

(Effect same as if we sort everyone by balance and get 10 people from the middle).

HowTo?

```
auto it = v.begin() + v.size() / 2 - 5;
```

HowTo?

```
auto it = v.begin() + v.size() / 2 - 5;

const auto f = [](const auto& v1, const auto& v2) {

    return v1.balance() < v2.balance();

};
```

HowTo?

```
auto it = v.begin() + v.size() / 2 - 5;
const auto f = [](const auto& v1, const auto& v2) {
    return v1.balance() < v2.balance();
};

std::nth_element(v.begin(), it, v.end(), f);
```

HowTo?

```
auto it = v.begin() + v.size() / 2 - 5;
const auto f = [](const auto& v1, const auto& v2) {
    return v1.balance() < v2.balance();
};

std::nth_element(v.begin(), it, v.end(), f);
std::partial_sort(it + 1, it + 10, v.end(), f);
```

O(N*log(10)) vs O(N*log(N))

N	N*log(10) $N+(N/2-1)*\log_2(9)$	N*log(N)	N*log(N) – N*log(10)
10	33	33	0
16	53	64	11
512	1,701	4,608	2,907
16,384	54,426	229,376	174,950
524,288	1,741,647	9,961,472	8,219,825
16,777,216	55,732,705	402,653,184	346,920,479

How to force the compiler to solve the previous task?

???

```
template <class RandomAccessIterator>
void nth_element(RandomAccessIterator beg, RandomAccessIterator mid,
RandomAccessIterator end)
    [[expects: beg <= mid]]    [[expects: mid <= end]]
    [[ensures: ??? ]]
;
template <class RandomAccessIterator>
void partial_sort(RandomAccessIterator beg, RandomAccessIterator mid,
RandomAccessIterator end)
    [[expects: beg <= mid]]    [[expects: mid <= end]]
    [[ensures: ??? ]]
;
```

Contracts and algorithms

Conclusion

Contracts – in development feature of the C++ language that allows to specify the function domain:

Conclusion

- Contracts – in development feature of the C++ language that allows to specify the function domain:
  - Unified syntax

Conclusion

- Contracts – in development feature of the C++ language that allows to specify the function domain:
  - Unified syntax
  - Start of the long way to formal verification

Conclusion

- Contracts – in development feature of the C++ language that allows to specify the function domain:
  - Unified syntax
  - Start of the long way to formal verification
  - Optimization hints

Conclusion

- Contracts – in development feature of the C++ language that allows to specify the function domain:
  - Unified syntax
  - Start of the long way to formal verification
  - Optimization hints
  - Problems

Conclusion

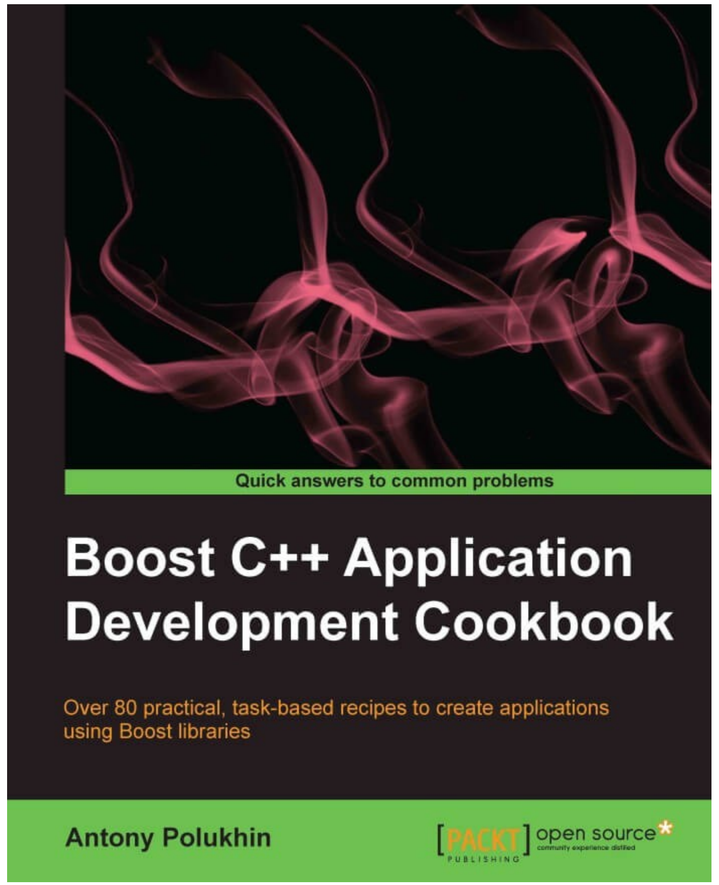
- Contracts – in development feature of the C++ language that allows to specify the function domain:
  - Unified syntax
  - Start of the long way to formal verification
  - Optimization hints
  - Problems
  - New horizons for research :-)

Thank you! Questions?

https://stdcpp.ru

C++

РГ21 C++ РОССИЯ



http://apolukhin.github.io/Boost-Cookbook