# Containers in Boost

- What containers has Boost to offer?
- How do they differ from STL containers?
- How do I know which one to use?
- How do I use them?
- Where do I find more information?

Boris Schäling, boris@highscore.de

Meeting C++, Düsseldorf, 9 November 2013

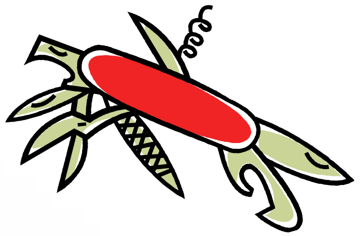
# Overview

This presentation covers the following 13 libraries which are more or less ordered by importance*):

Boost.Multiindex
Boost.Bimap
Boost.Container
Boost.Intrusive
Boost.PointerContainer
Boost.CircularBuffer
Boost.Lockfree
Boost.PropertyTree
Boost.DynamicBitset
Boost.Multiarray
Boost.Heap
Boost.Array
Boost.Unordered

*) Very subjective

# Boost.Multiindex



Create new containers which provide multiple interfaces to lookup items

- One container – multiple interfaces (indexes)
- No need to split up types for associative indexes

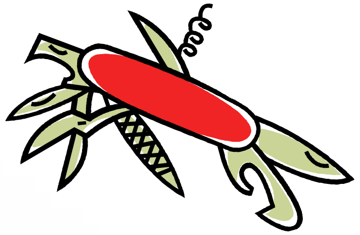
## Header files

```
#include <boost/multi_index_container.hpp>
#include <boost/multi_index/....hpp>
```

## Namespace

```
using namespace boost::multi_index;
```

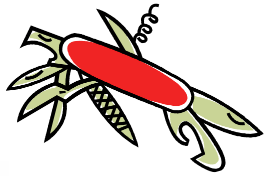
# Boost.Multiindex

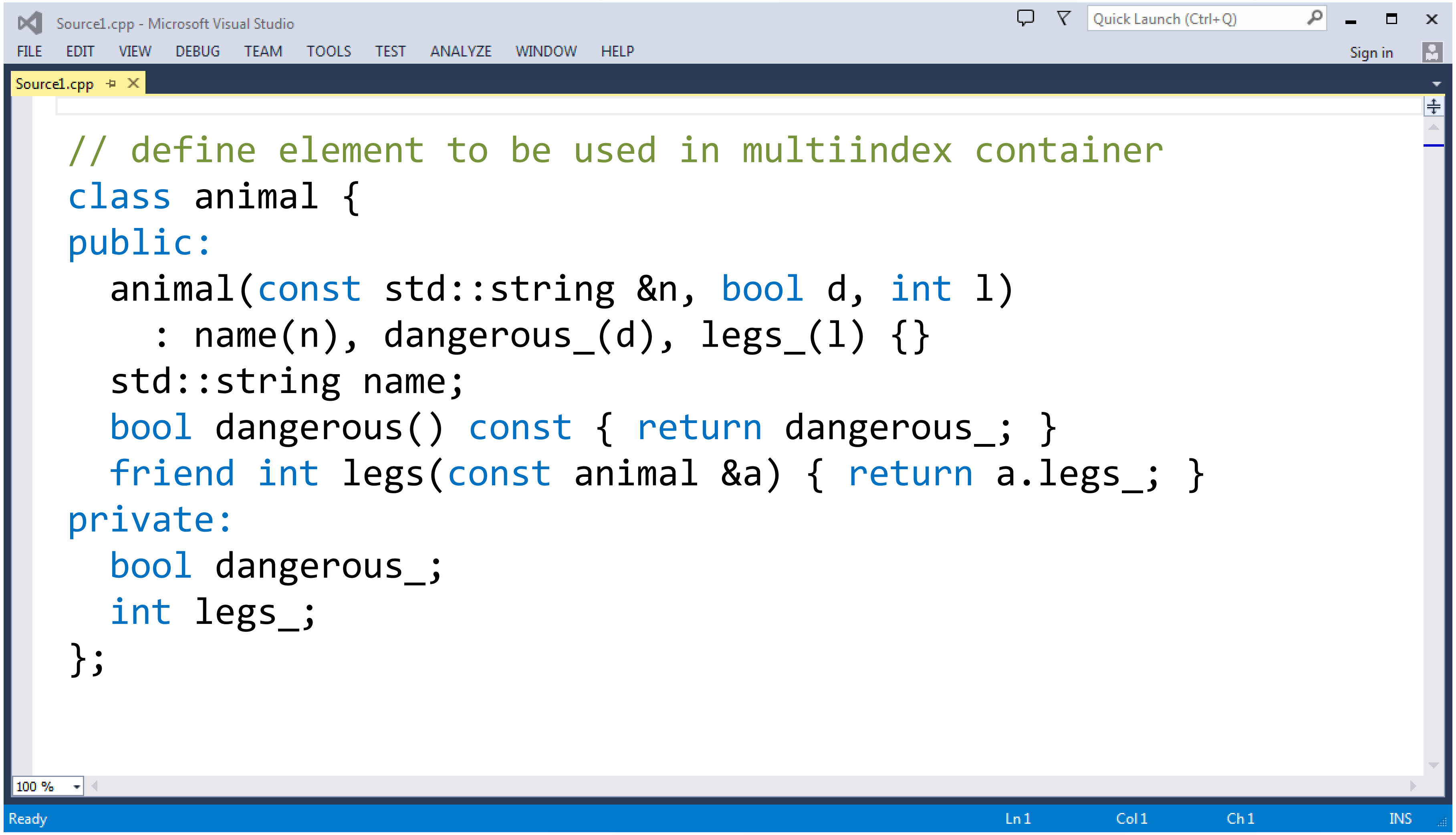


| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ✓ |
| Can be serialized with Boost.Serialization | ✓ |
| Can be shared with Boost.Interprocess | ✓ |

Since Boost 1.32.0

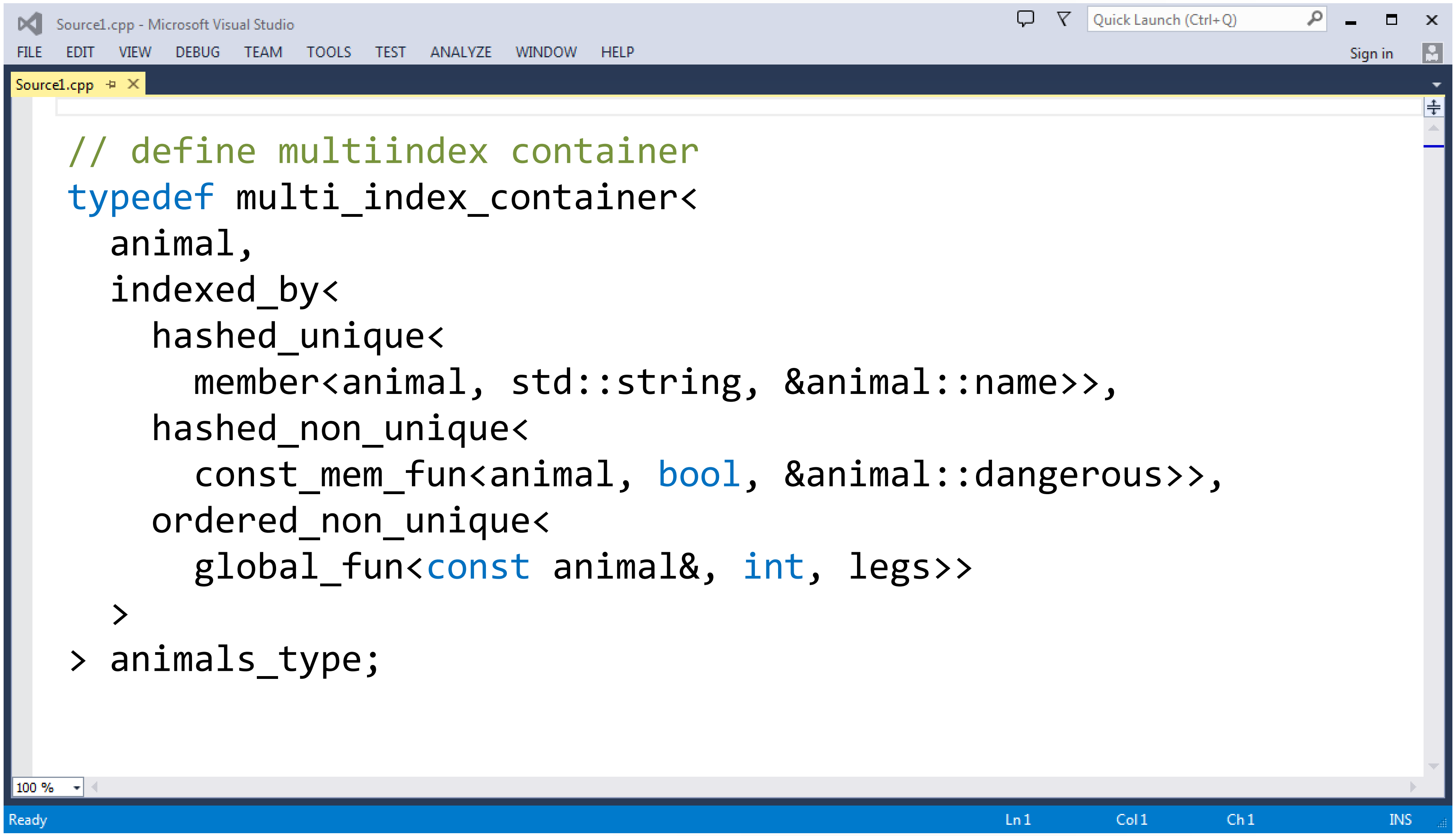
# Boost.Multiindex




Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

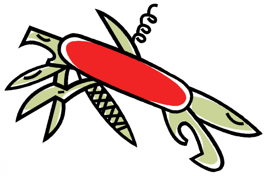
```cpp
// define element to be used in multiindex container
class animal {
public:
  animal(const std::string &n, bool d, int l)
    : name(n), dangerous_(d), legs_(l) {}
  std::string name;
  bool dangerous() const { return dangerous_; }
  friend int legs(const animal &a) { return a.legs_; }
private:
  bool dangerous_;
  int legs_;
};
```

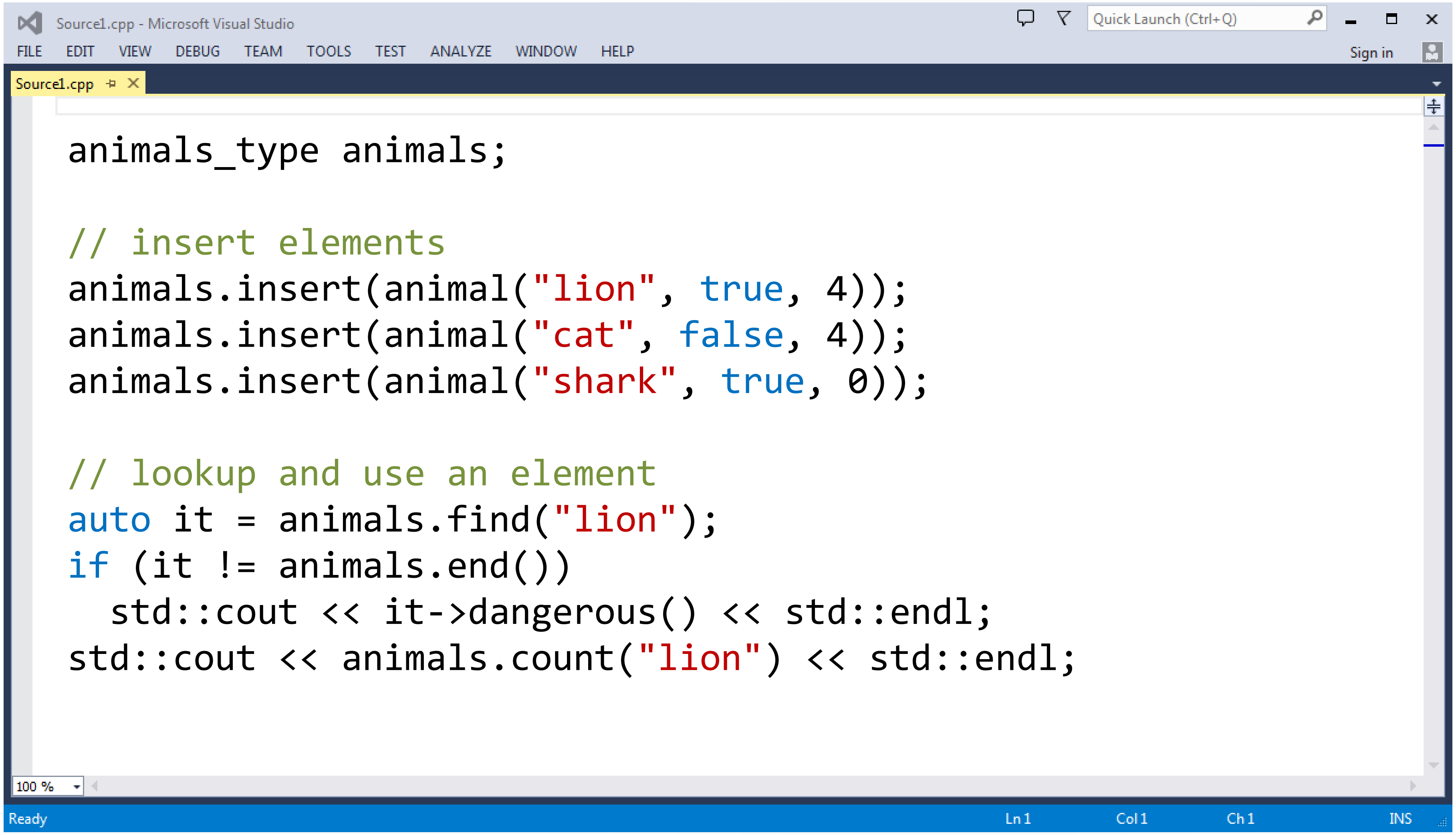
# Boost.Multiindex


Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// define multiindex container
typedef multi_index_container<
  animal,
  indexed_by<
    hashed_unique<
      member<animal, std::string, &animal::name>>,
    hashed_non_unique<
      const_mem_fun<animal, bool, &animal::dangerous>>,
    ordered_non_unique<
      global_fun<const animal&, int, legs>>
  >
> animals_type;
```

# Boost.Multiindex



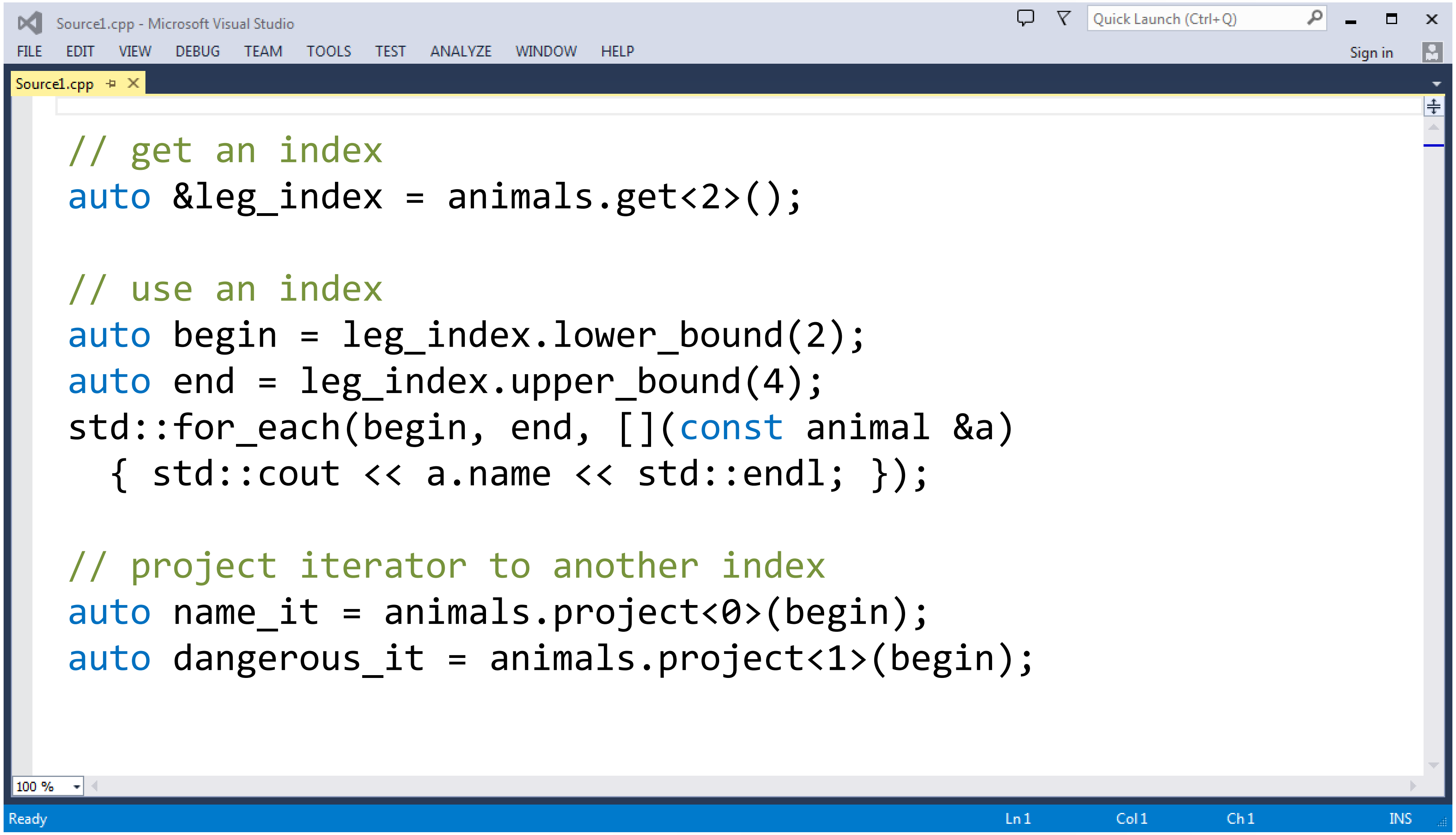

Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
animals_type animals;

// insert elements
animals.insert(animal("lion", true, 4));
animals.insert(animal("cat", false, 4));
animals.insert(animal("shark", true, 0));

// lookup and use an element
auto it = animals.find("lion");
if (it != animals.end())
  std::cout << it->dangerous() << std::endl;
std::cout << animals.count("lion") << std::endl;
```

# Boost.Multiindex

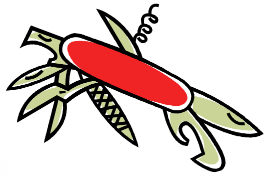

Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

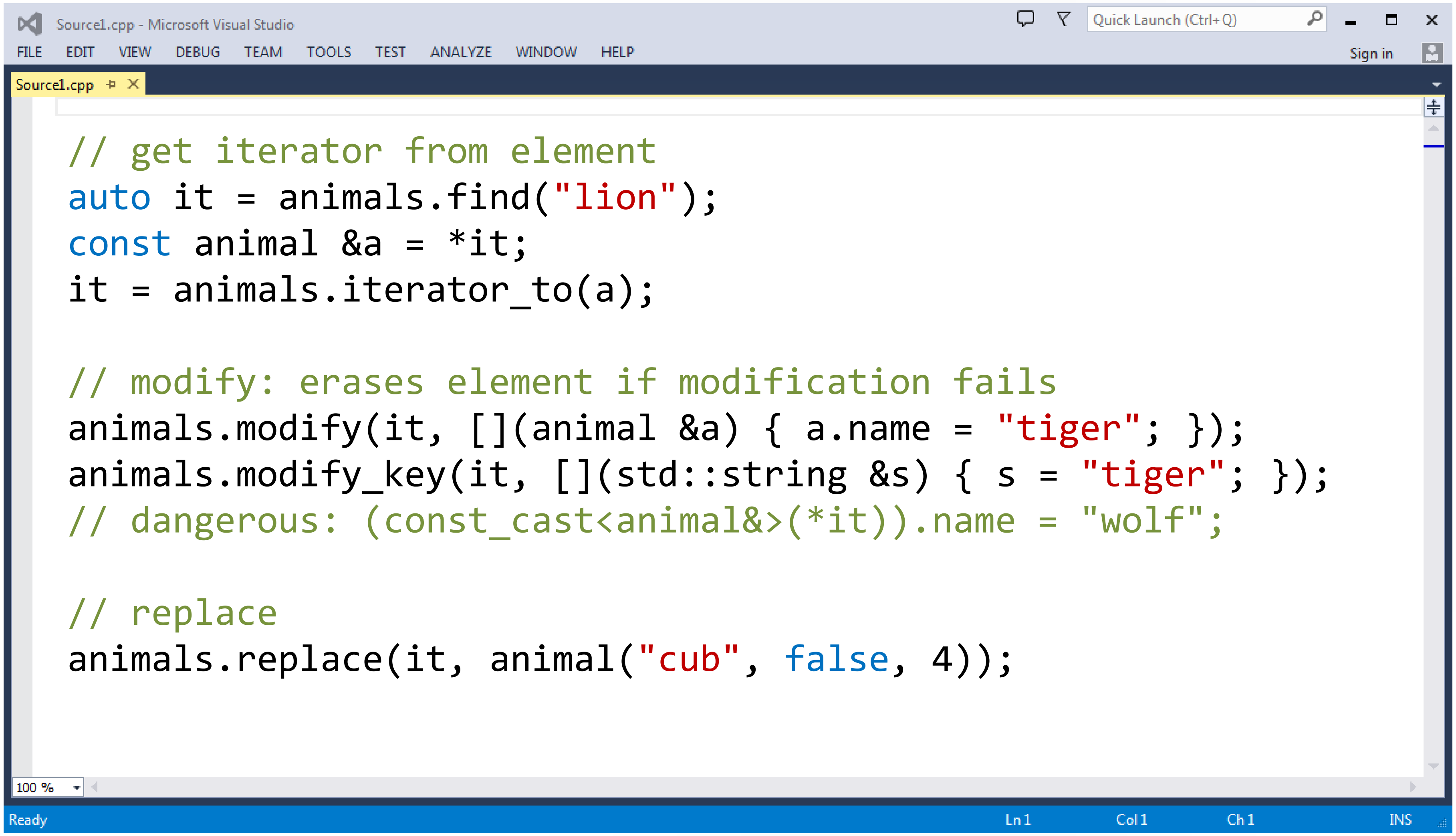
```
// get an index
auto &leg_index = animals.get<2>();

// use an index
auto begin = leg_index.lower_bound(2);
auto end = leg_index.upper_bound(4);
std::for_each(begin, end, [](const animal &a)
  { std::cout << a.name << std::endl; });

// project iterator to another index
auto name_it = animals.project<0>(begin);
auto dangerous_it = animals.project<1>(begin);
```

# Boost.Multiindex



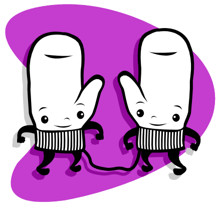

Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// get iterator from element
auto it = animals.find("lion");
const animal &a = *it;
it = animals.iterator_to(a);

// modify: erases element if modification fails
animals.modify(it, [](animal &a) { a.name = "tiger"; });
animals.modify_key(it, [](std::string &s) { s = "tiger"; });
// dangerous: (const_cast<animal&>(*it)).name = "wolf";

// replace
animals.replace(it, animal("cub", false, 4));
```

## Boost.Bimap



A std::map-like container which supports lookups from both sides

- Lookup data from left or right side
- Iterate over pair-relations

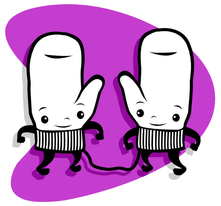
### Header files

```
#include <boost/bimap.hpp>
#include <boost/bimap/....hpp>
```

### Namespace

```
using namespace boost::bimaps;
```

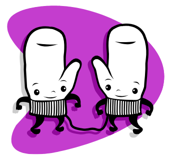
## Boost.Bimap

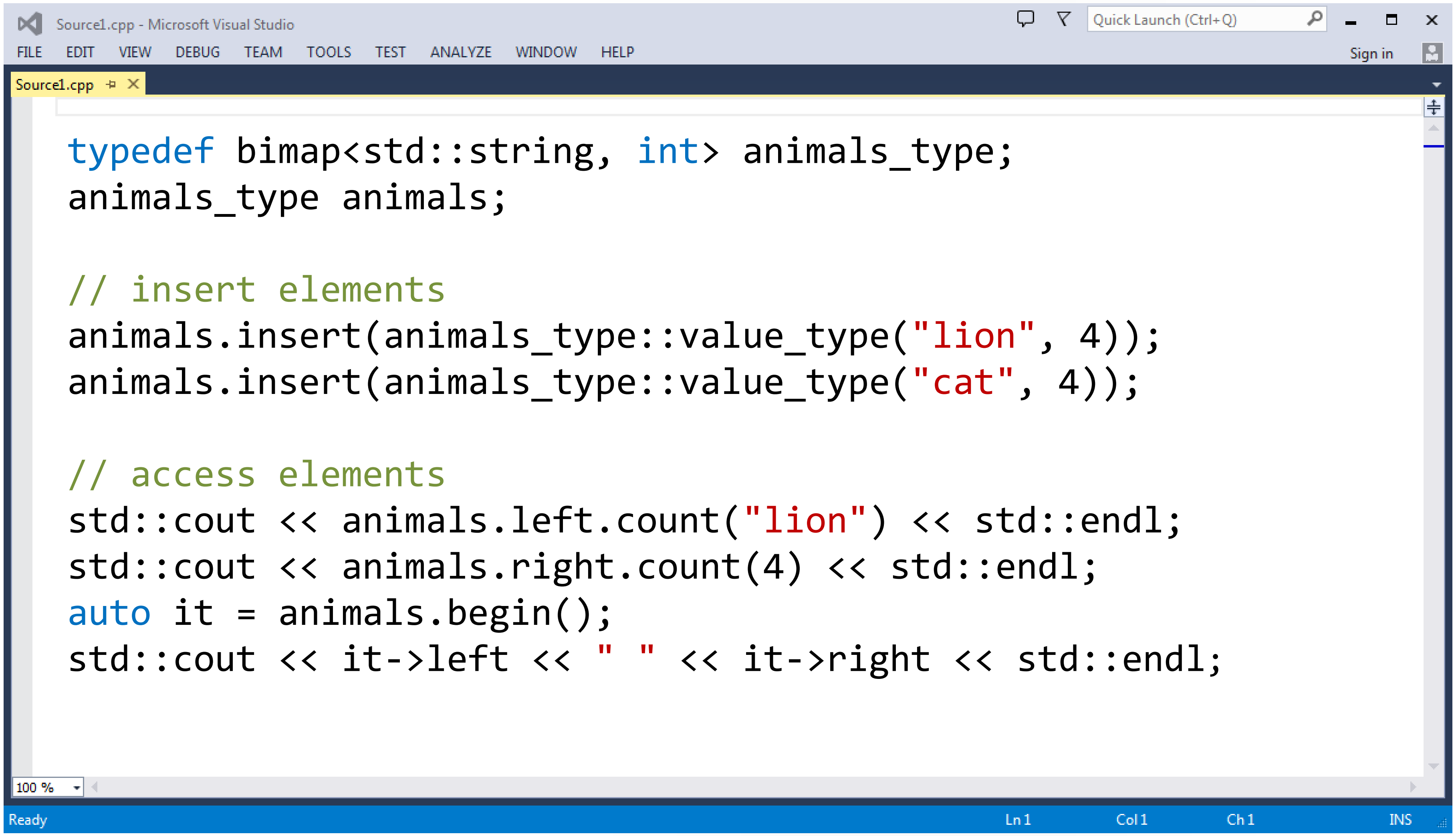


| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ✓ |
| Can be serialized with Boost.Serialization | ✓ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.35.0

# Boost.Bimap



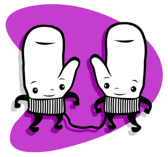

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

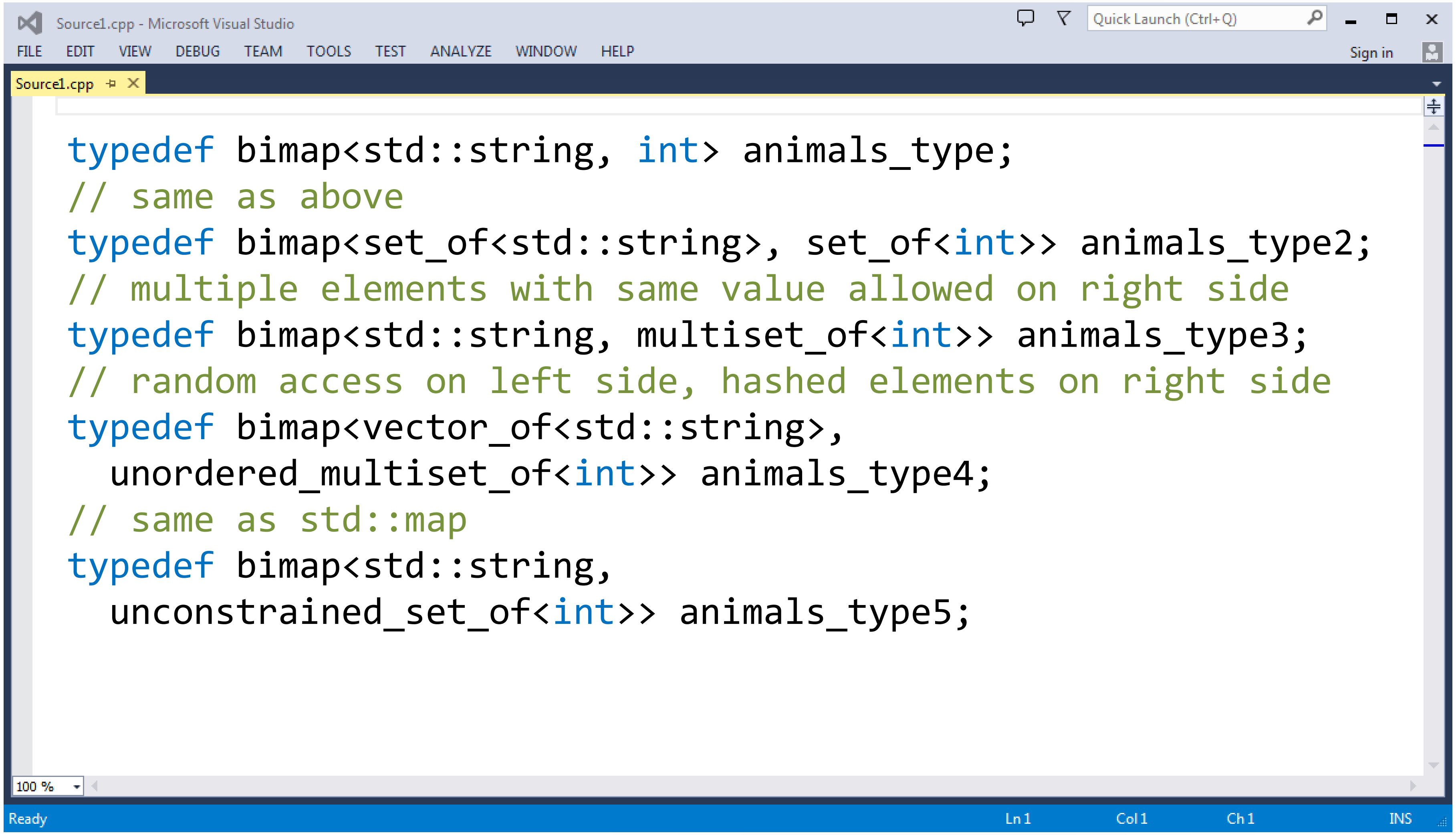
```cpp
typedef bimap<std::string, int> animals_type;
animals_type animals;

// insert elements
animals.insert(animals_type::value_type("lion", 4));
animals.insert(animals_type::value_type("cat", 4));

// access elements
std::cout << animals.left.count("lion") << std::endl;
std::cout << animals.right.count(4) << std::endl;
auto it = animals.begin();
std::cout << it->left << " " << it->right << std::endl;
```

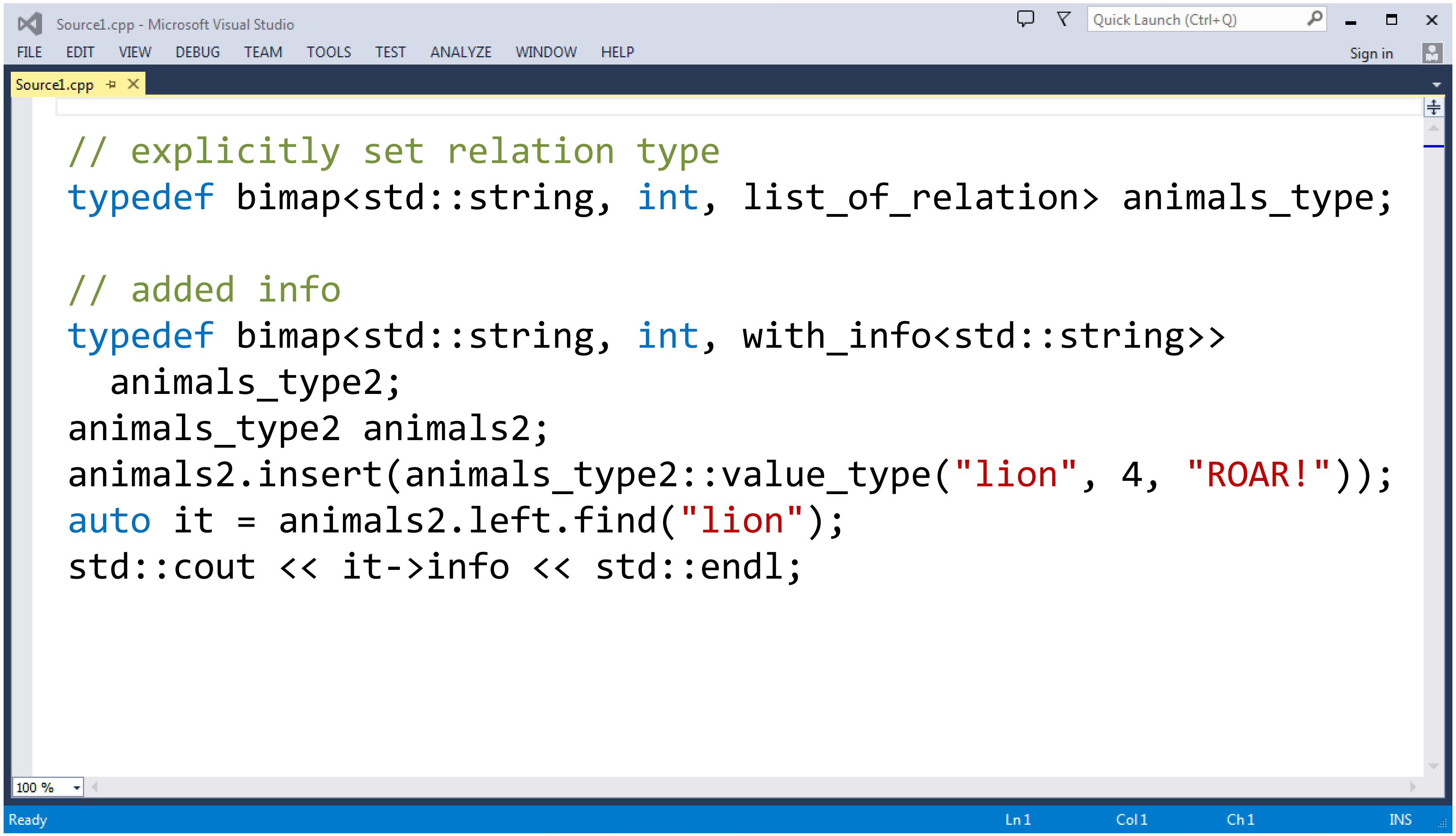
# Boost.Bimap




Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
typedef bimap<std::string, int> animals_type;
// same as above
typedef bimap<set_of<std::string>, set_of<int>> animals_type2;
// multiple elements with same value allowed on right side
typedef bimap<std::string, multiset_of<int>> animals_type3;
// random access on left side, hashed elements on right side
typedef bimap<vector_of<std::string>,
  unordered_multiset_of<int>> animals_type4;
// same as std::map
typedef bimap<std::string,
  unconstrained_set_of<int>> animals_type5;
```

# Boost.Bimap

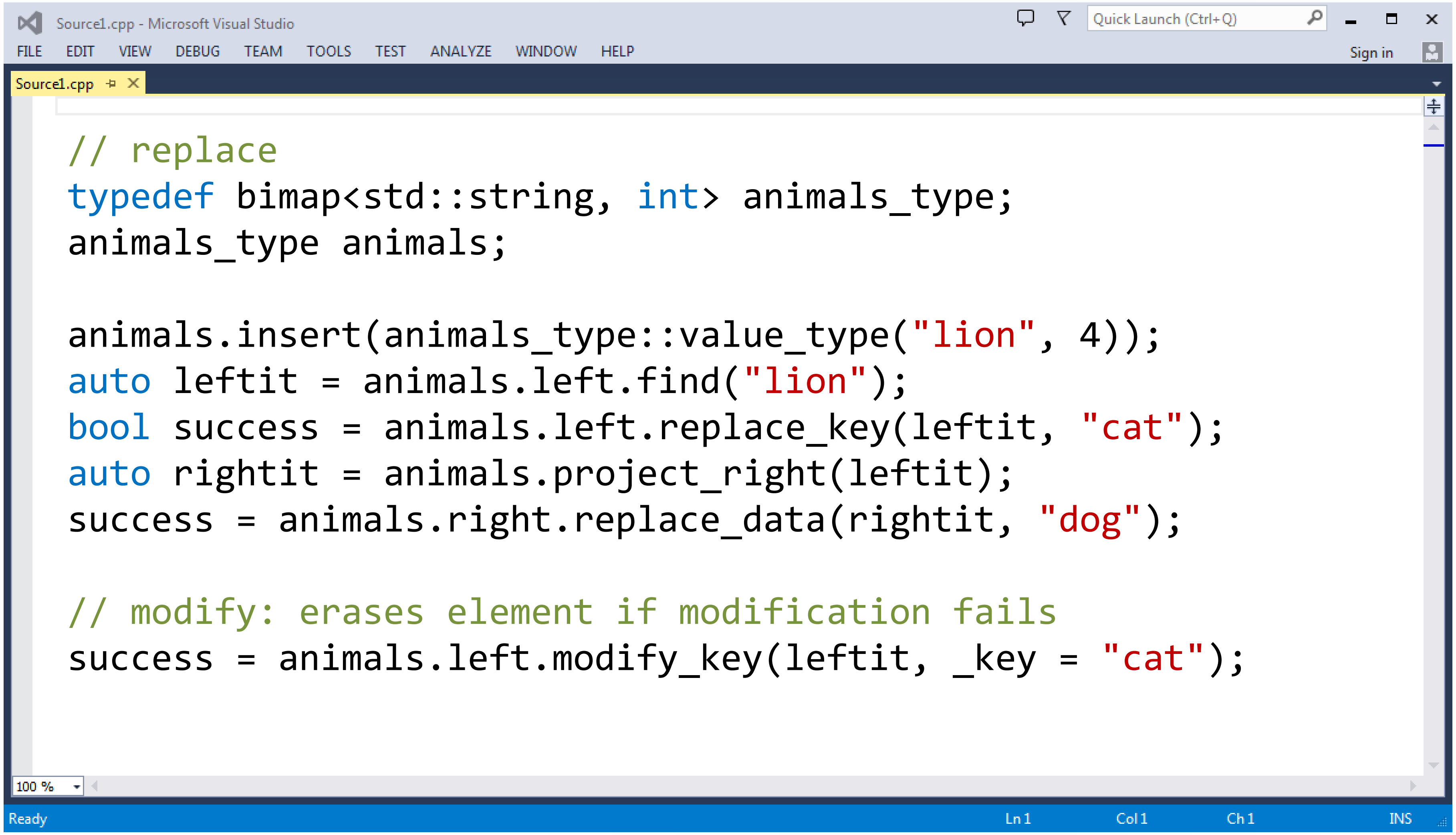


Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// explicitly set relation type
typedef bimap<std::string, int, list_of_relation> animals_type;

// added info
typedef bimap<std::string, int, with_info<std::string>>
  animals_type2;
animals_type2 animals2;
animals2.insert(animals_type2::value_type("lion", 4, "ROAR!"));
auto it = animals2.left.find("lion");
std::cout << it->info << std::endl;
```

# Boost.Bimap

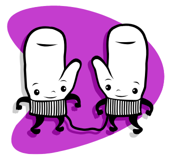

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

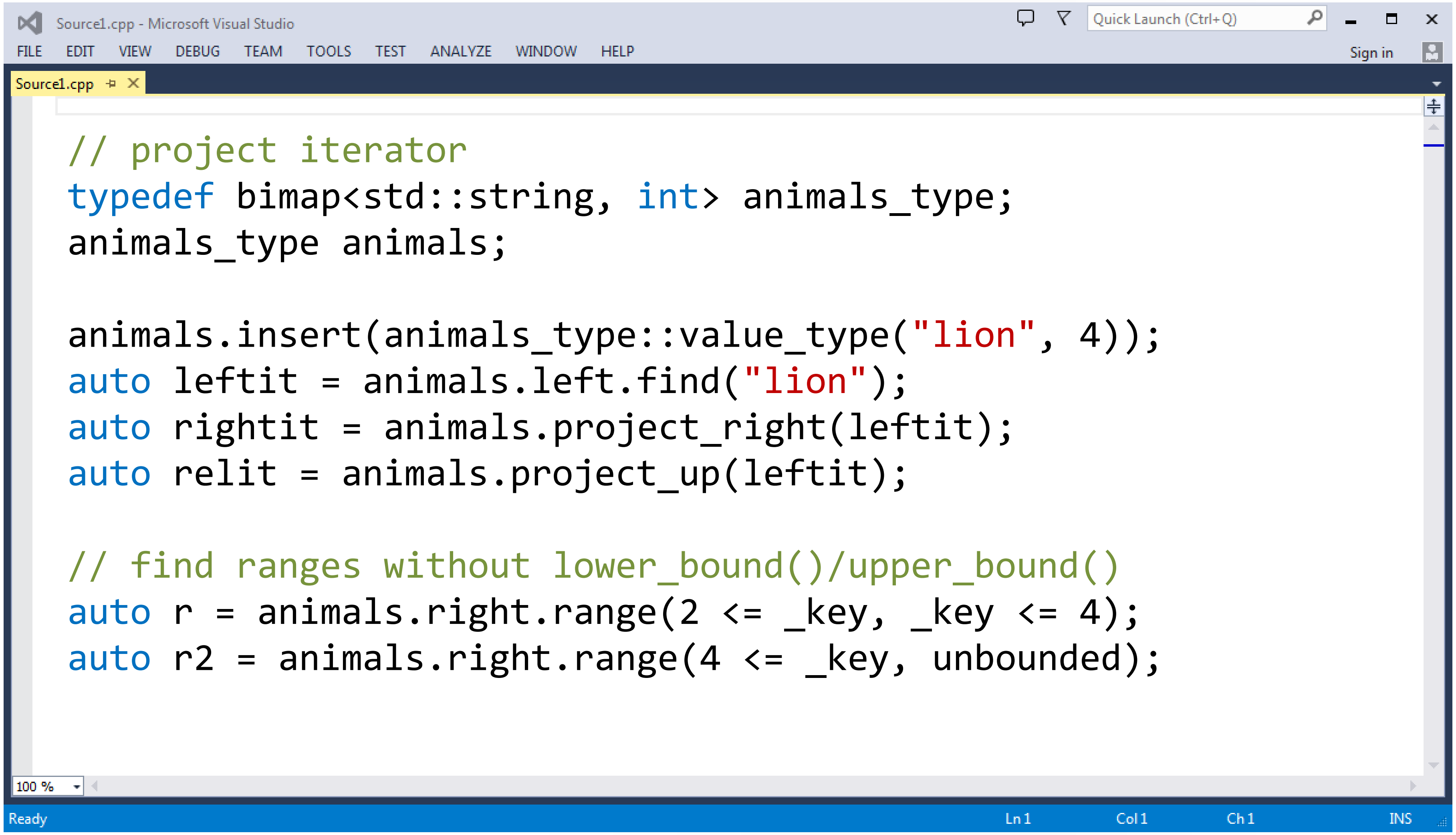
```cpp
// replace
typedef bimap<std::string, int> animals_type;
animals_type animals;

animals.insert(animals_type::value_type("lion", 4));
auto leftit = animals.left.find("lion");
bool success = animals.left.replace_key(leftit, "cat");
auto rightit = animals.project_right(leftit);
success = animals.right.replace_data(rightit, "dog");

// modify: erases element if modification fails
success = animals.left.modify_key(leftit, _key = "cat");
```

# Boost.Bimap




Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// project iterator
typedef bimap<std::string, int> animals_type;
animals_type animals;

animals.insert(animals_type::value_type("lion", 4));
auto leftit = animals.left.find("lion");
auto rightit = animals.project_right(leftit);
auto relit = animals.project_up(leftit);

// find ranges without lower_bound()/upper_bound()
auto r = animals.right.range(2 <= _key, _key <= 4);
auto r2 = animals.right.range(4 <= _key, unbounded);
```

## Boost.Container

Same containers as in the C++ standard library but with some extra comfort

- Recursive containers possible
- Boost has stable_vector, static_vector, flat_set, flat_map, slist and small string optimization

### Header files

```
#include <boost/container/....hpp>
```

### Namespace

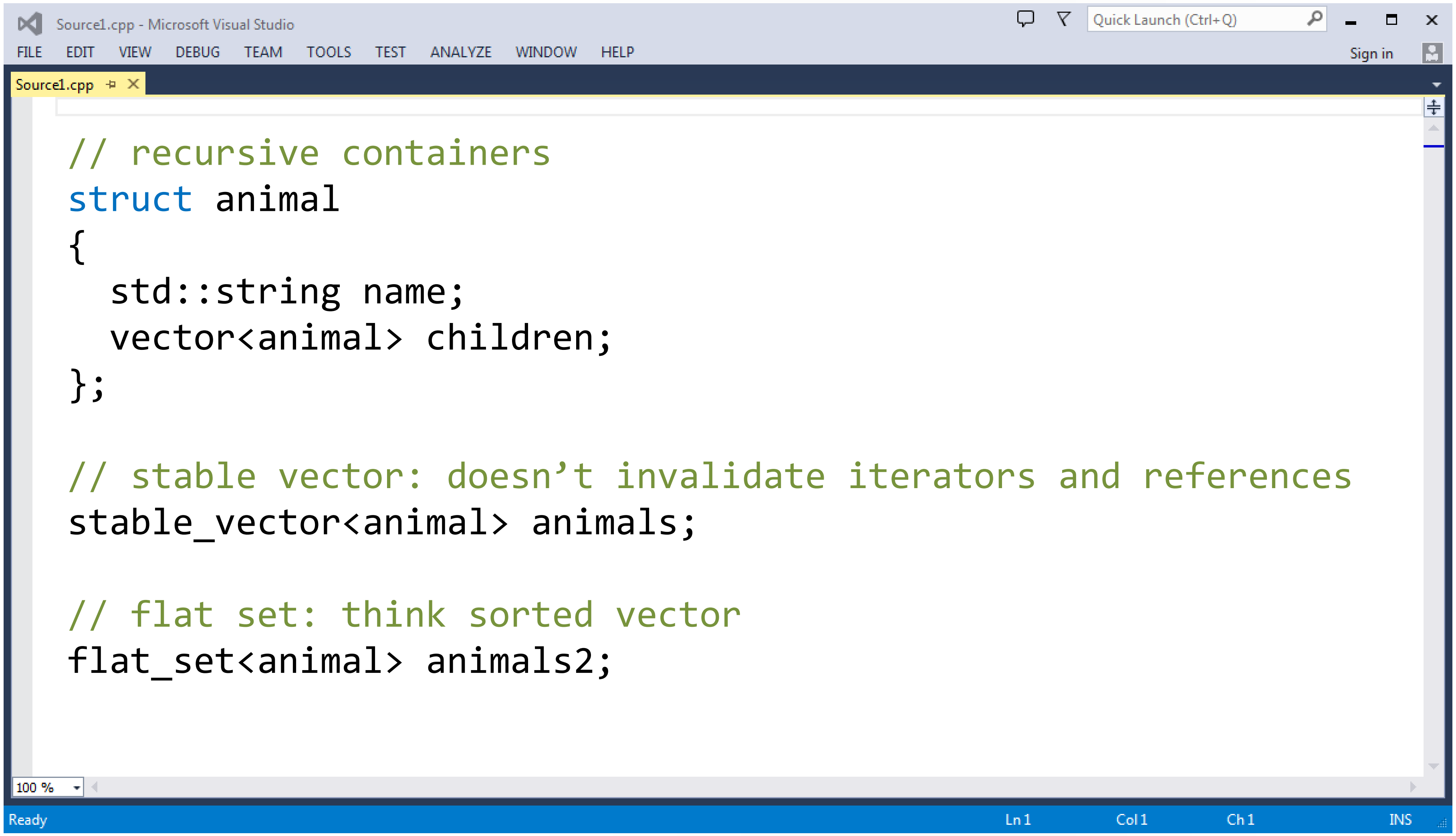
```
using namespace boost::container;
```

# Boost.Container

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ✓ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ☐ |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ✓ |

Since Boost 1.48.0

# Boost.Container


Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```
// recursive containers
struct animal
{
  std::string name;
  vector<animal> children;
};

// stable vector: doesn’t invalidate iterators and references
stable_vector<animal> animals;

// flat set: think sorted vector
flat_set<animal> animals2;
```

# Boost.Intrusive

Containers which don‘t store copies of objects but original objects

- Lifetime of elements must be managed by user
- Types must be setup to be used in containers
- Lots of containers provided

## Header files

#include <boost/intrusive/….hpp>

## Namespace

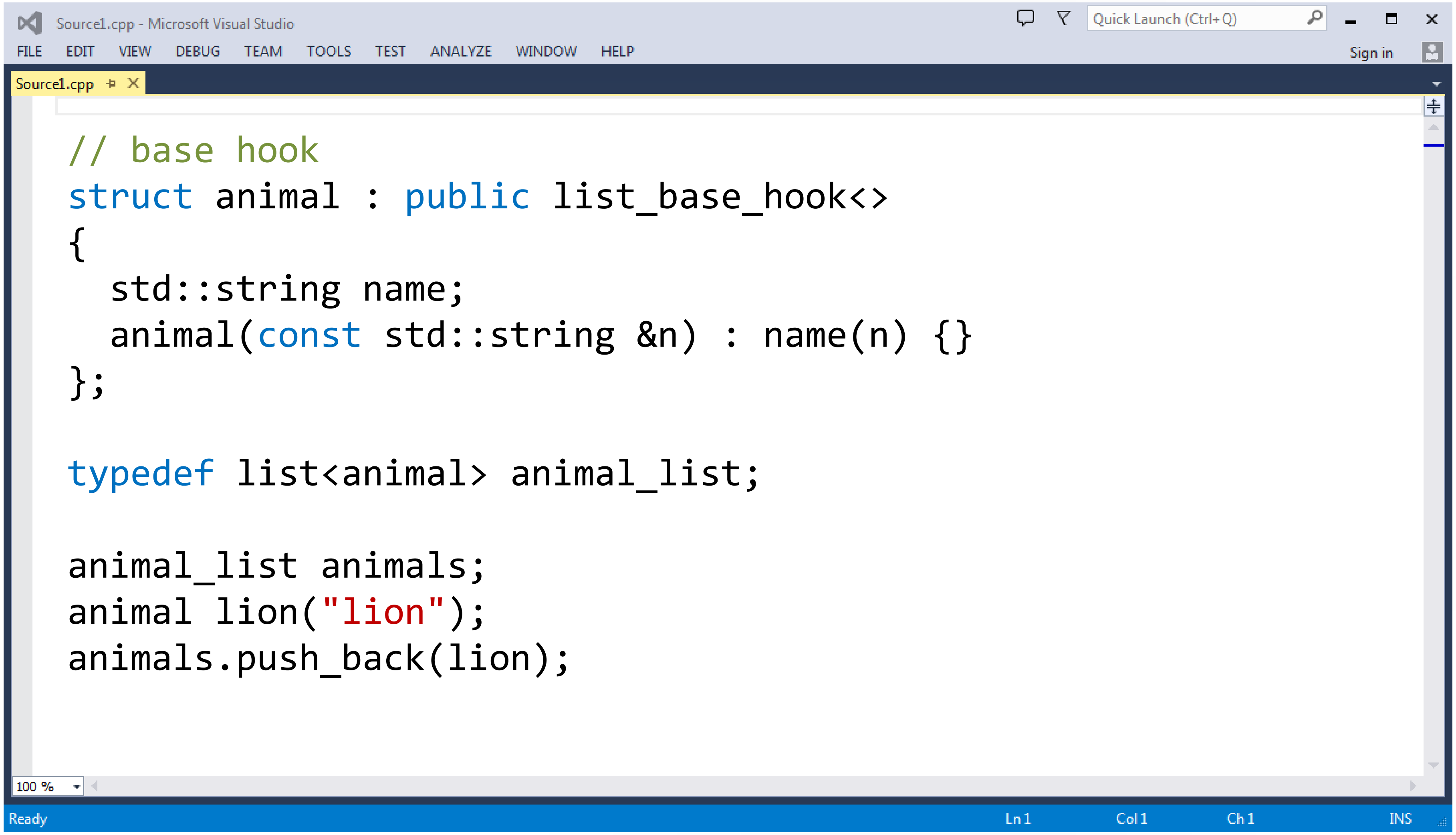
using namespace boost::intrusive;

# Boost.Intrusive

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ✓ |
| Fixed-size | ☐ |
| Owns elements | ☐ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ✓ |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.35.0

# Boost.Intrusive

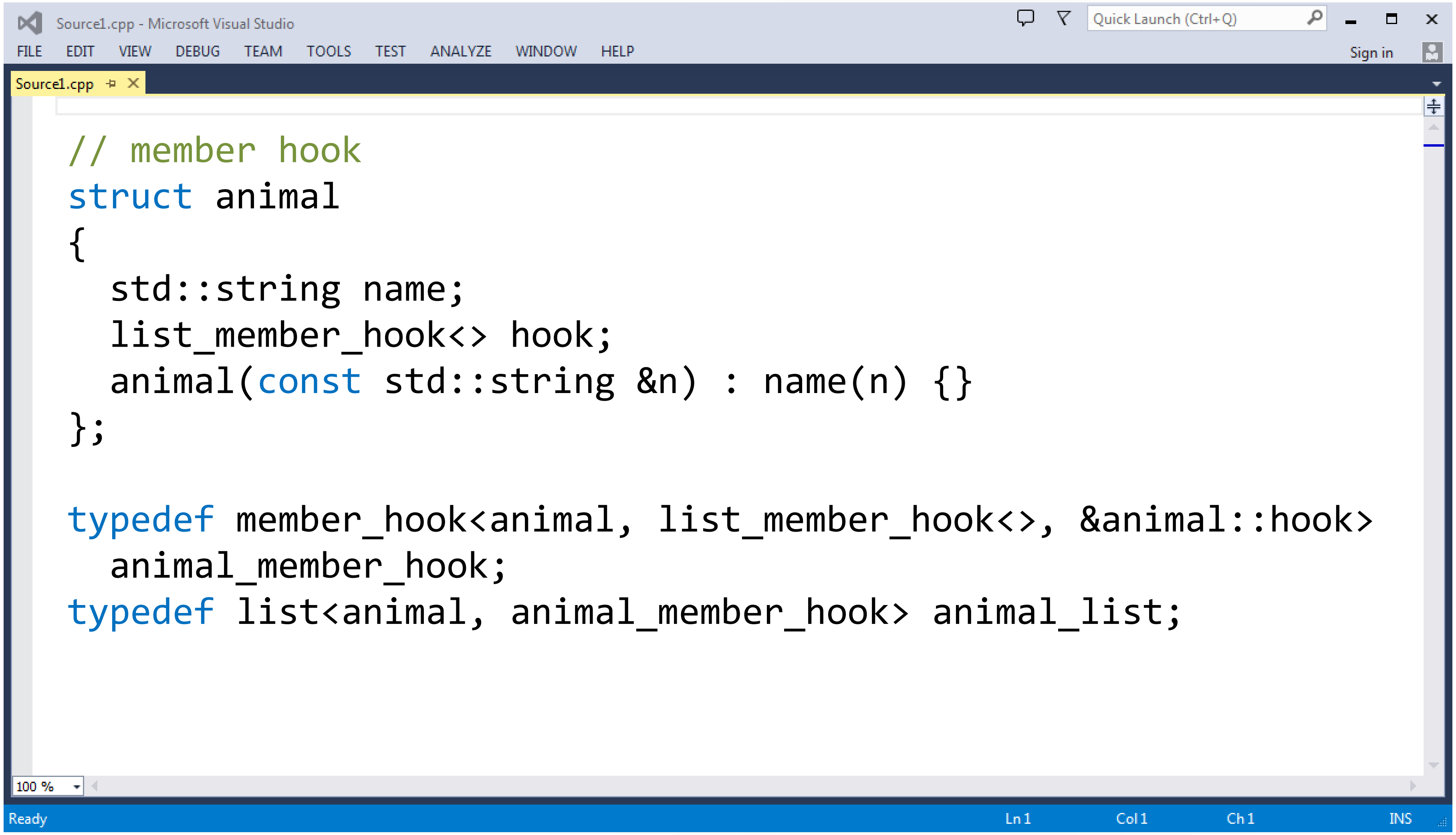

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// base hook
struct animal : public list_base_hook<>
{
  std::string name;
  animal(const std::string &n) : name(n) {}
};

typedef list<animal> animal_list;

animal_list animals;
animal lion("lion");
animals.push_back(lion);
```

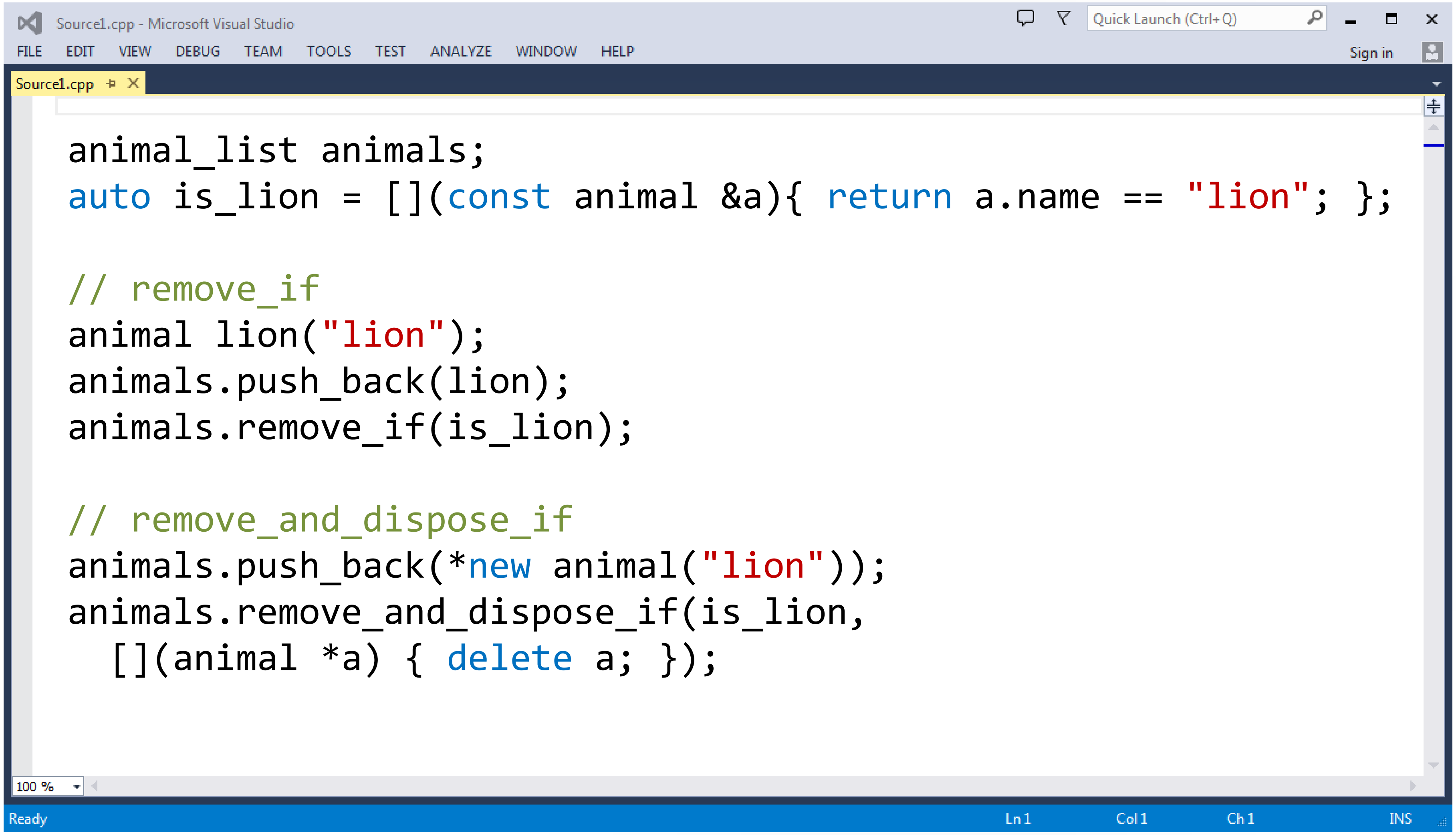
# Boost.Intrusive


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// member hook
struct animal
{
  std::string name;
  list_member_hook<> hook;
  animal(const std::string &n) : name(n) {}
};

typedef member_hook<animal, list_member_hook<>, &animal::hook>
  animal_member_hook;
typedef list<animal, animal_member_hook> animal_list;
```

# Boost.Intrusive

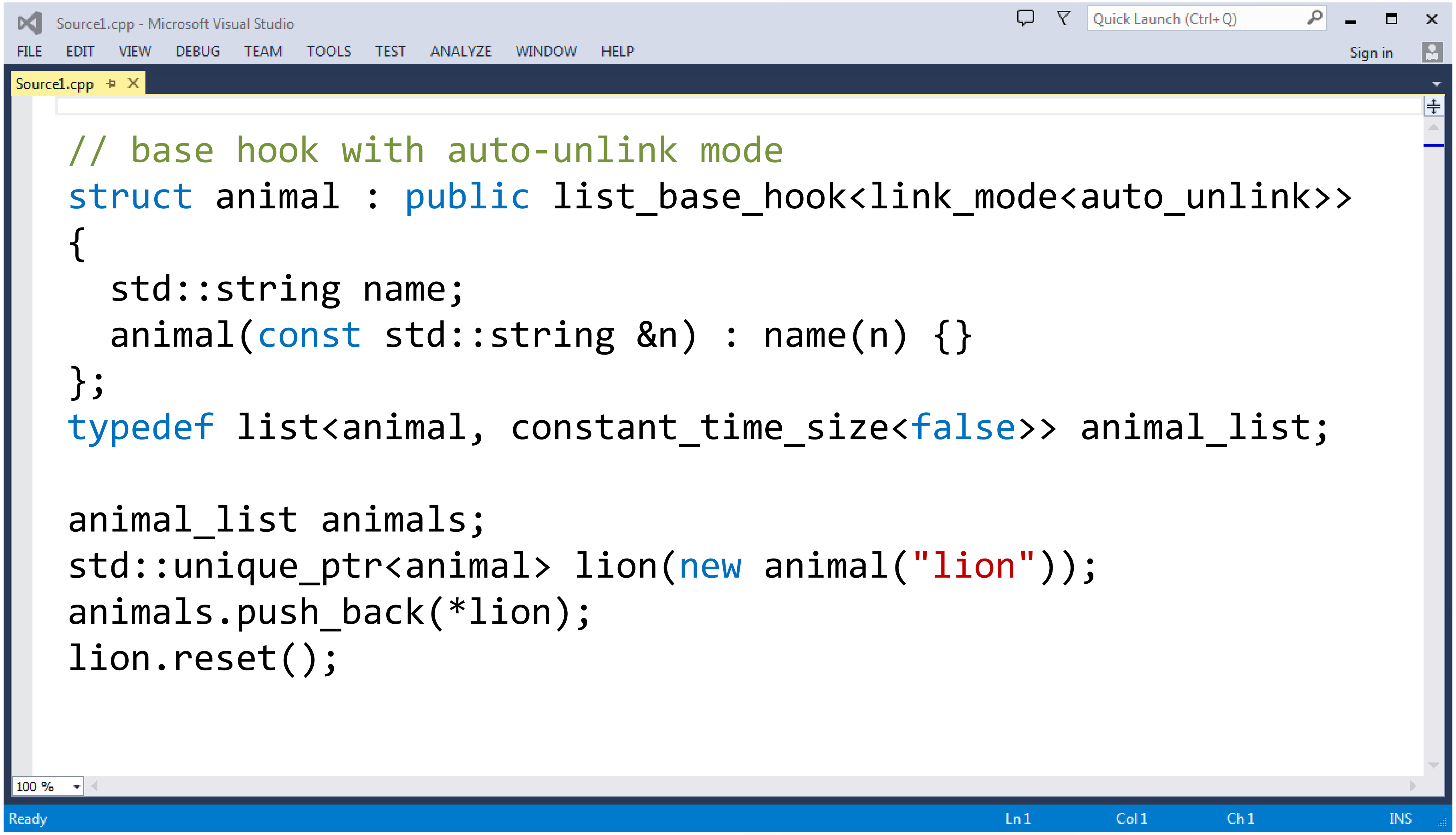

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
animal_list animals;
auto is_lion = [](const animal &a){ return a.name == "lion"; };

// remove_if
animal lion("lion");
animals.push_back(lion);
animals.remove_if(is_lion);

// remove_and_dispose_if
animals.push_back(*new animal("lion"));
animals.remove_and_dispose_if(is_lion,
  [](animal *a) { delete a; });
```

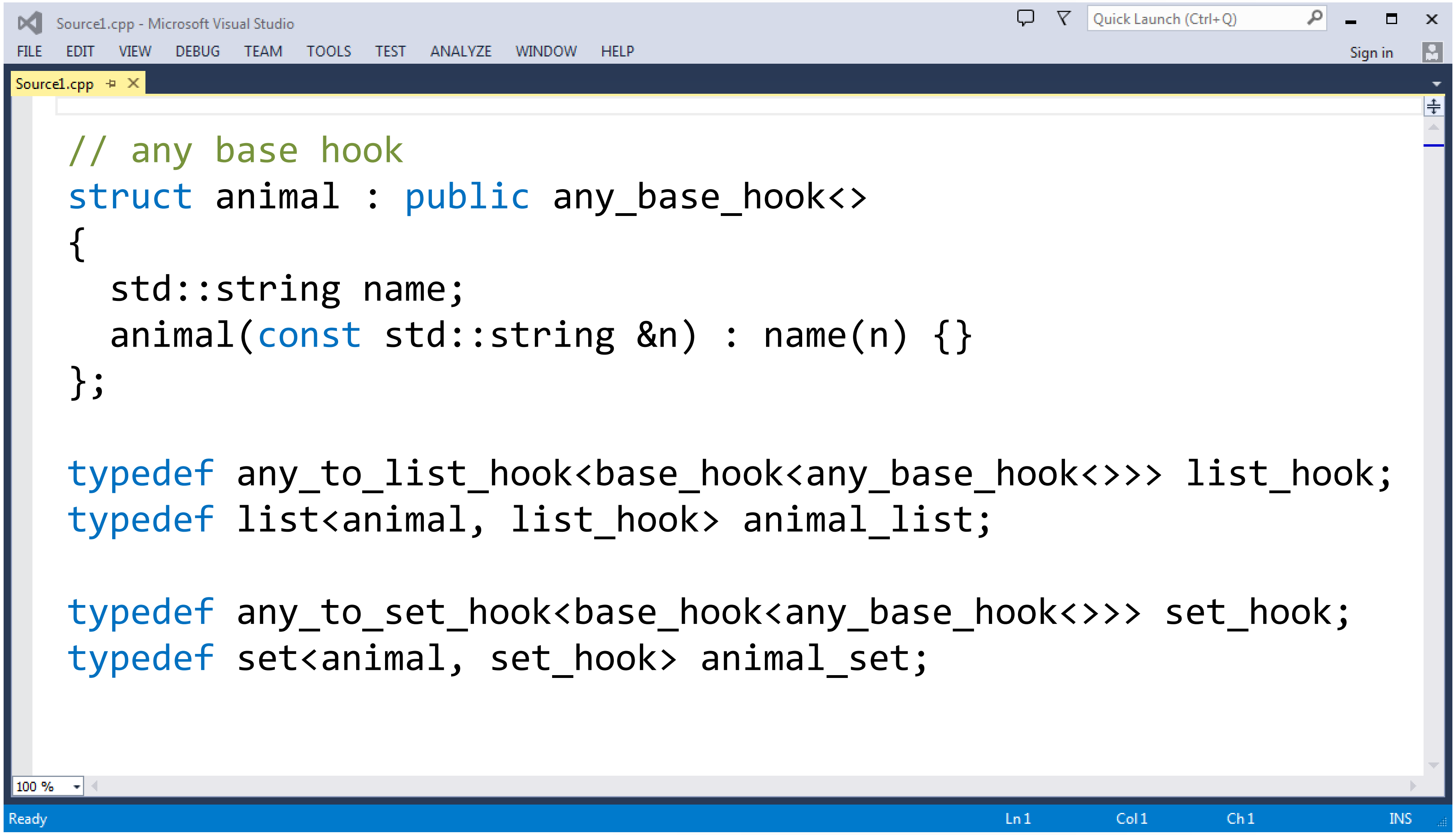
# Boost.Intrusive


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// base hook with auto-unlink mode
struct animal : public list_base_hook<link_mode<auto_unlink>>
{
  std::string name;
  animal(const std::string &n) : name(n) {}
};
typedef list<animal, constant_time_size<false>> animal_list;

animal_list animals;
std::unique_ptr<animal> lion(new animal("lion"));
animals.push_back(*lion);
lion.reset();
```

# Boost.Intrusive


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// any base hook
struct animal : public any_base_hook<>
{
  std::string name;
  animal(const std::string &n) : name(n) {}
};

typedef any_to_list_hook<base_hook<any_base_hook<>>> list_hook;
typedef list<animal, list_hook> animal_list;

typedef any_to_set_hook<base_hook<any_base_hook<>>> set_hook;
typedef set<animal, set_hook> animal_set;
```

## Boost.PointerContainer

STL-like containers which manage dynamically allocated objects

- Similar to std::vector<std::unique_ptr>>
- Iterators point to objects directly
- Insert iterators provided

### Header files

#include <boost/ptr_container/....hpp>

### Namespace

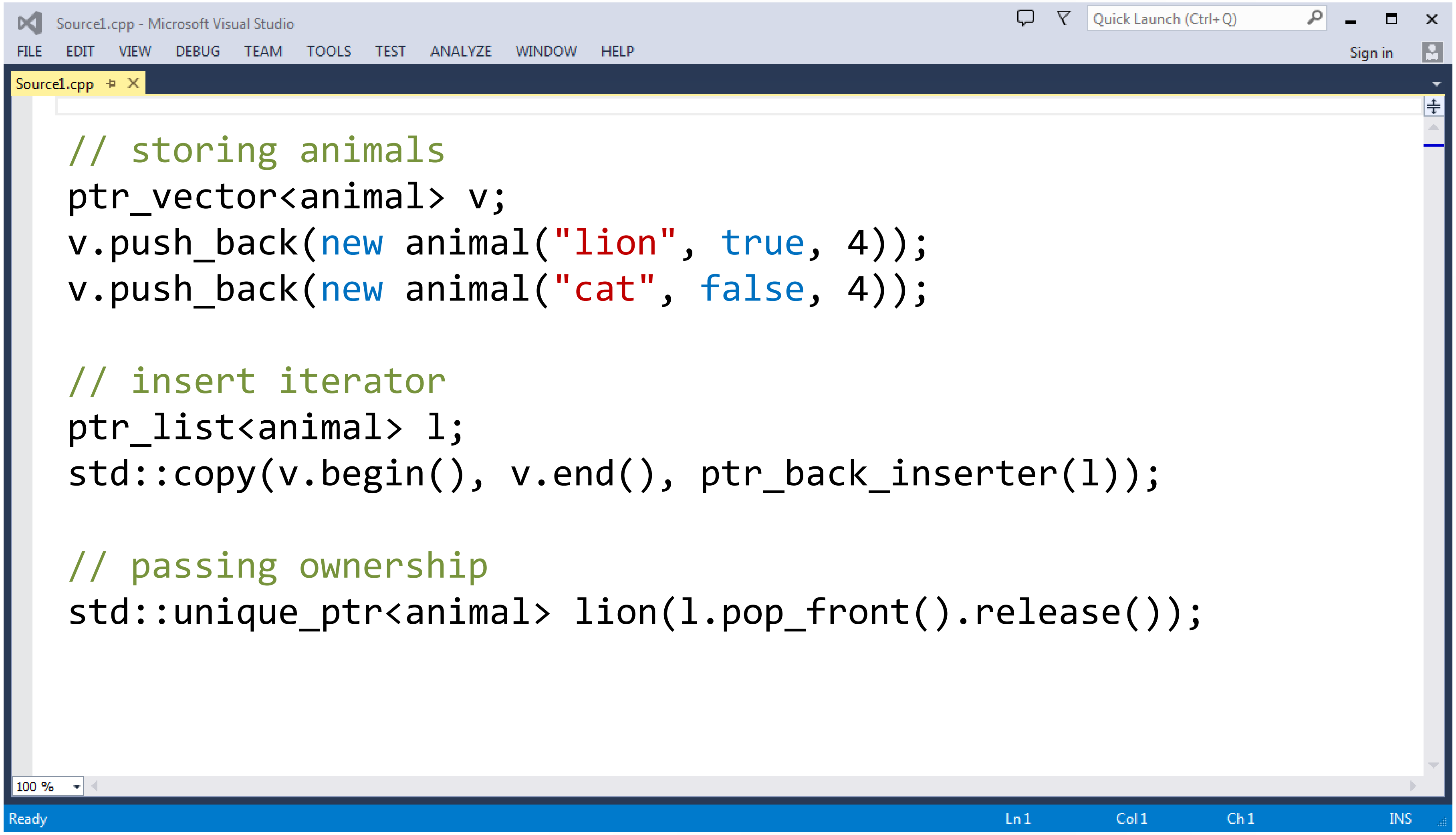
using namespace boost;

# Boost.PointerContainer

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ☐ |
| Can be serialized with Boost.Serialization | ✓ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.33.0

# Boost.PointerContainer


Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// storing animals
ptr_vector<animal> v;
v.push_back(new animal("lion", true, 4));
v.push_back(new animal("cat", false, 4));

// insert iterator
ptr_list<animal> l;
std::copy(v.begin(), v.end(), ptr_back_inserter(l));

// passing ownership
std::unique_ptr<animal> lion(l.pop_front().release());
```

# Boost.CircularBuffer

A fixed-size container which overwrites elements if you keep on inserting more

- Overwriting is done through assignment
- Size is set at runtime
- Has begin and end iterators

## Header file

#include <boost/circular_buffer.hpp>

## Namespace

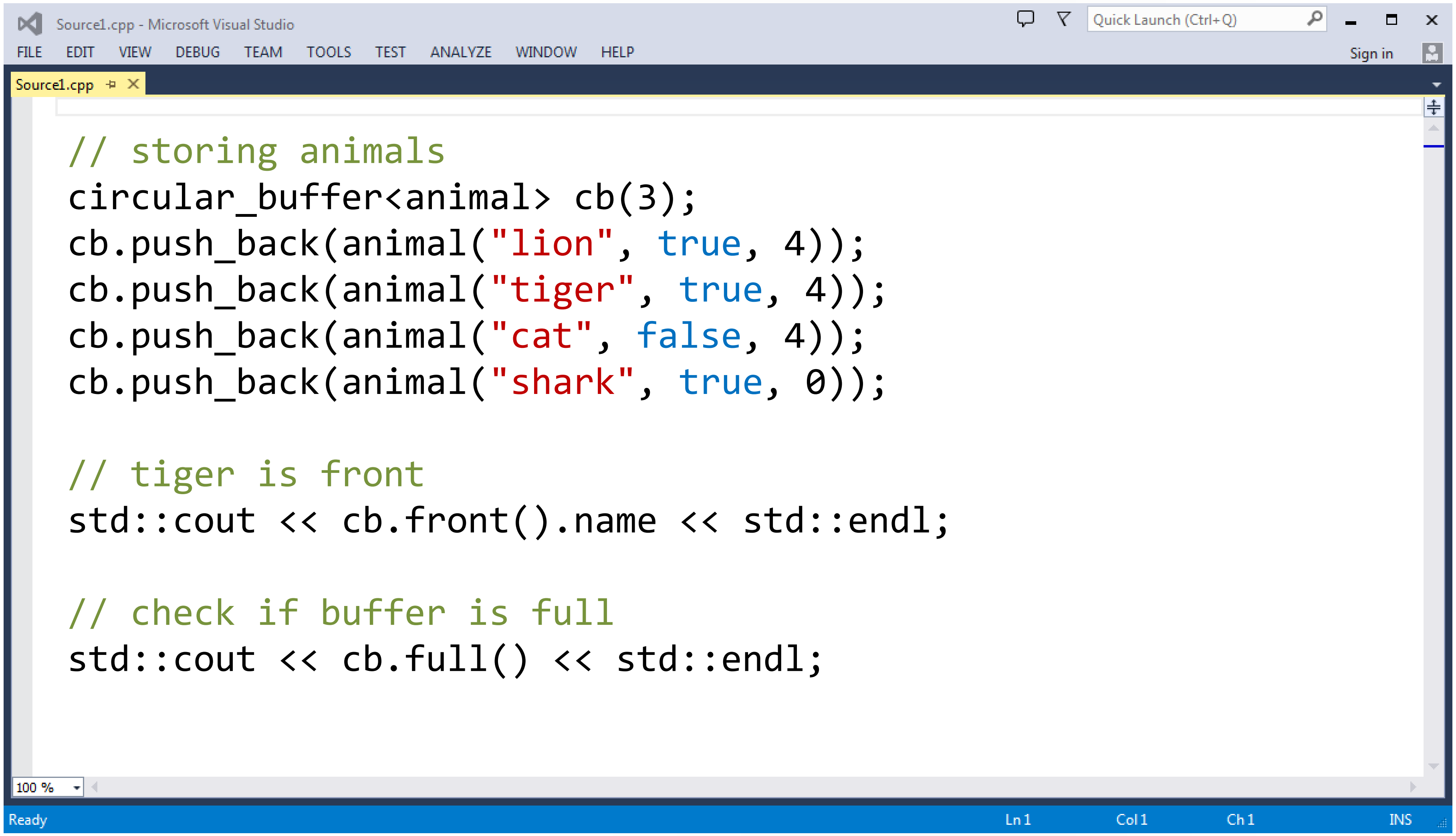
using namespace boost;

# Boost.CircularBuffer

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ▢ |
| Fixed-size | ✓ |
| Owns elements | ✓ |
| Thread-safe | ▢ |
| Validity of iterators and references preserved | ▢ |
| Can be serialized with Boost.Serialization | ▢ |
| Can be shared with Boost.Interprocess | ✓ |

Since Boost 1.35.0

# Boost.CircularBuffer

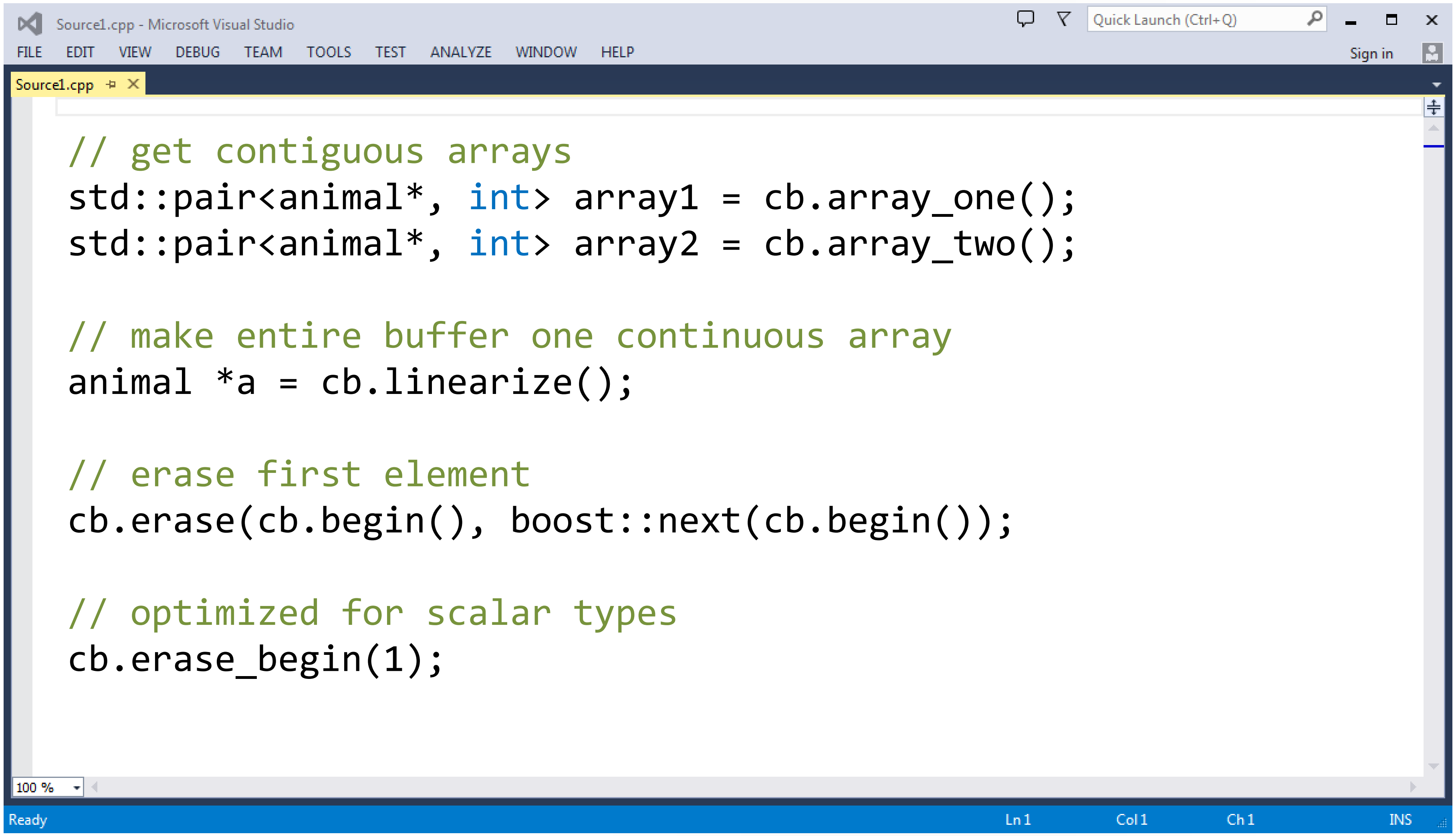

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// storing animals
circular_buffer<animal> cb(3);
cb.push_back(animal("lion", true, 4));
cb.push_back(animal("tiger", true, 4));
cb.push_back(animal("cat", false, 4));
cb.push_back(animal("shark", true, 0));

// tiger is front
std::cout << cb.front().name << std::endl;

// check if buffer is full
std::cout << cb.full() << std::endl;
```

# Boost.CircularBuffer


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// get contiguous arrays
std::pair<animal*, int> array1 = cb.array_one();
std::pair<animal*, int> array2 = cb.array_two();

// make entire buffer one continuous array
animal *a = cb.linearize();

// erase first element
cb.erase(cb.begin(), boost::next(cb.begin()));

// optimized for scalar types
cb.erase_begin(1);
```

# Boost.Lockfree

Provides a lock-free queue and a stack which can be concurrently modified in multiple threads

- Atomic operations
- Support for fixed size containers
- Multi and single producer/consumer use cases

## Header files

#include <boost/lockfree/....hpp>

## Namespace

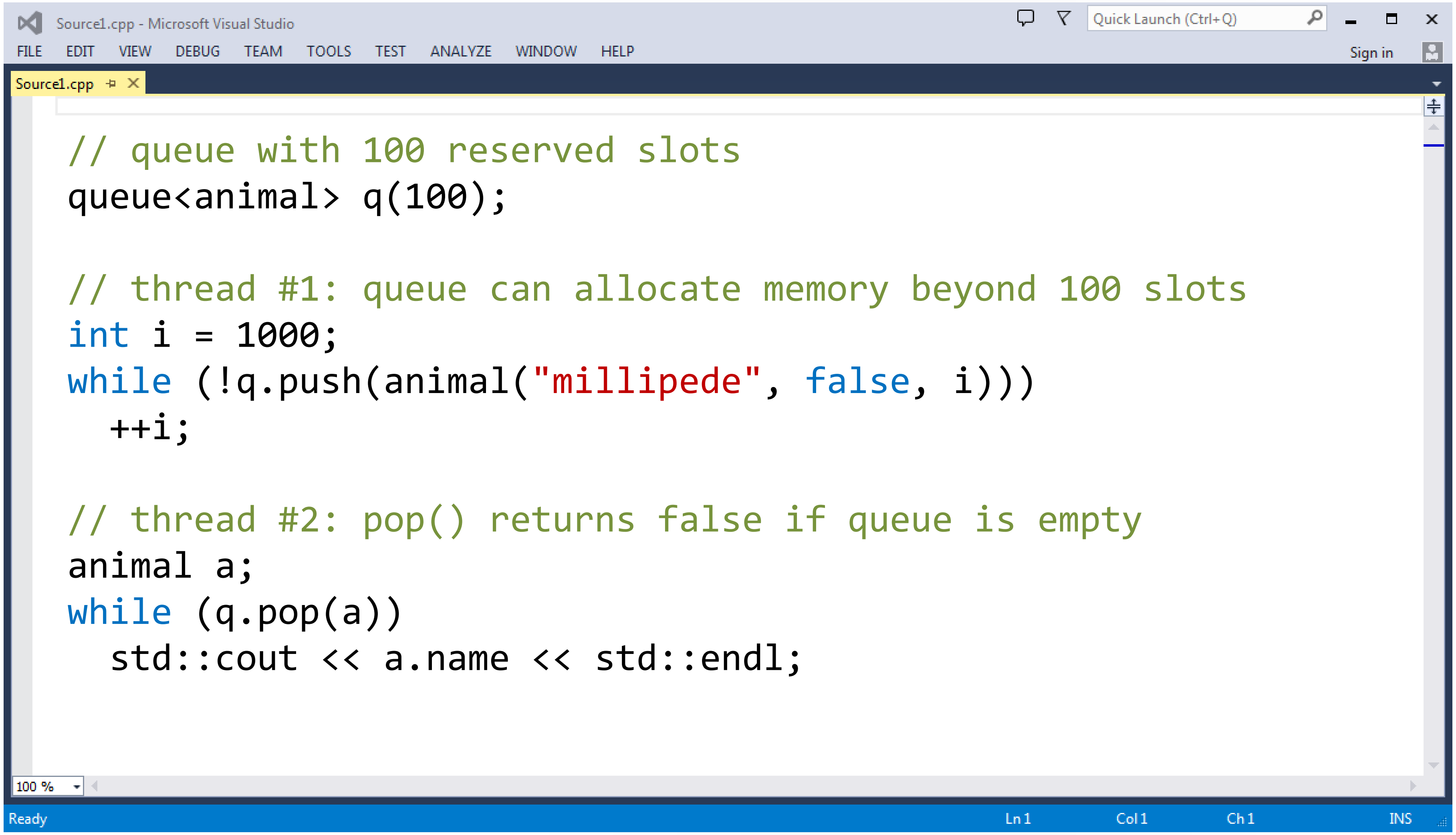
using namespace boost::lockfree;

# Boost.Lockfree

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ✓ |
| Validity of iterators and references preserved | ☐ |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ✓ |

Since Boost 1.53.0

# Boost.Lockfree

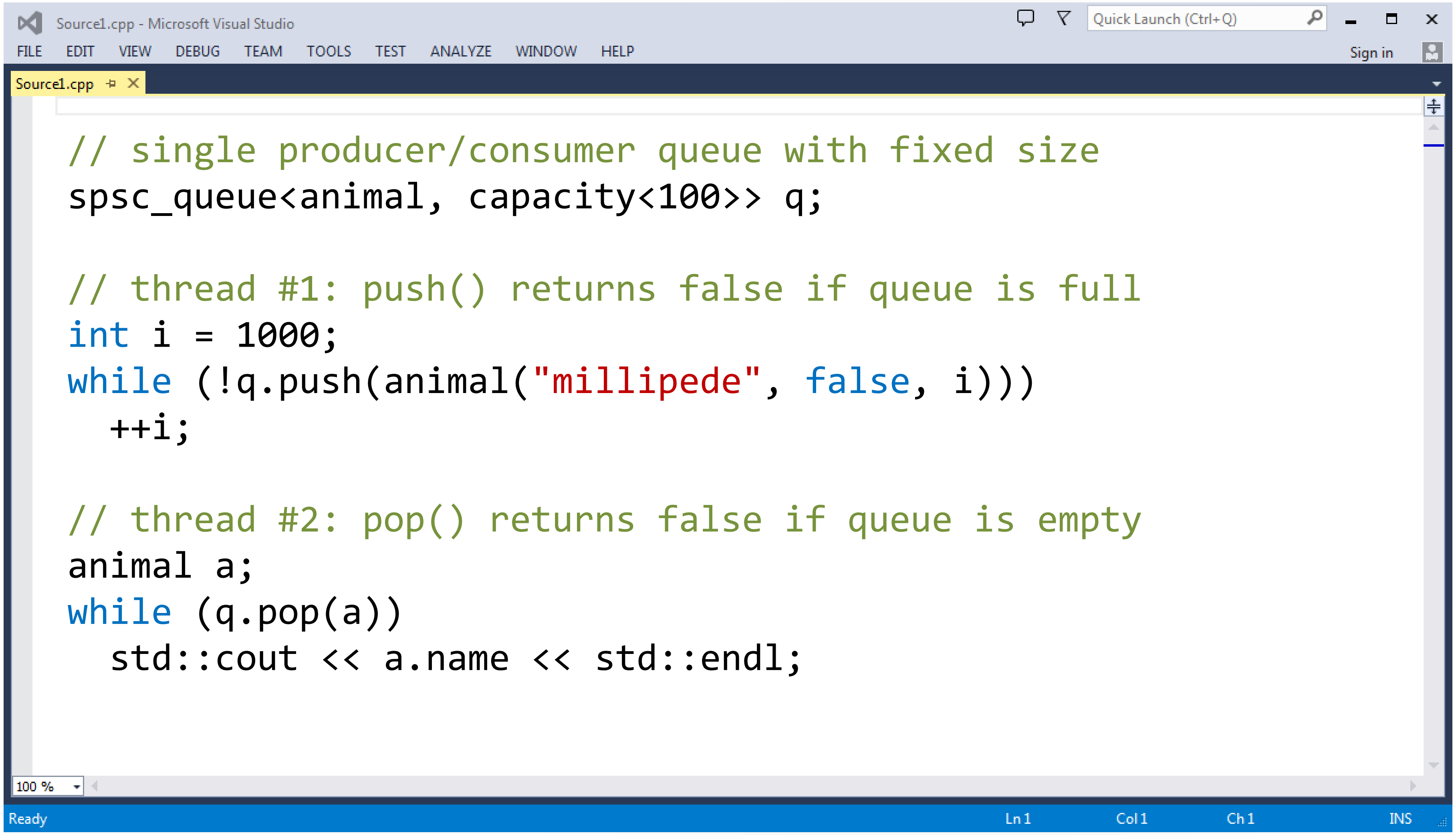

Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// queue with 100 reserved slots
queue<animal> q(100);

// thread #1: queue can allocate memory beyond 100 slots
int i = 1000;
while (!q.push(animal("millipede", false, i)))
  ++i;

// thread #2: pop() returns false if queue is empty
animal a;
while (q.pop(a))
  std::cout << a.name << std::endl;
```

# Boost.Lockfree


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// single producer/consumer queue with fixed size
spsc_queue<animal, capacity<100>> q;

// thread #1: push() returns false if queue is full
int i = 1000;
while (!q.push(animal("millipede", false, i)))
  ++i;

// thread #2: pop() returns false if queue is empty
animal a;
while (q.pop(a))
  std::cout << a.name << std::endl;
```

## Boost.PropertyTree

A tree container with key/value pairs which can be saved to and loaded from files

- Use for configuration data
- Supports XML, JSON and INI formats
- Supports keys to extract data from anywhere

### Header files

#include <boost/property_tree/....hpp>

### Namespace

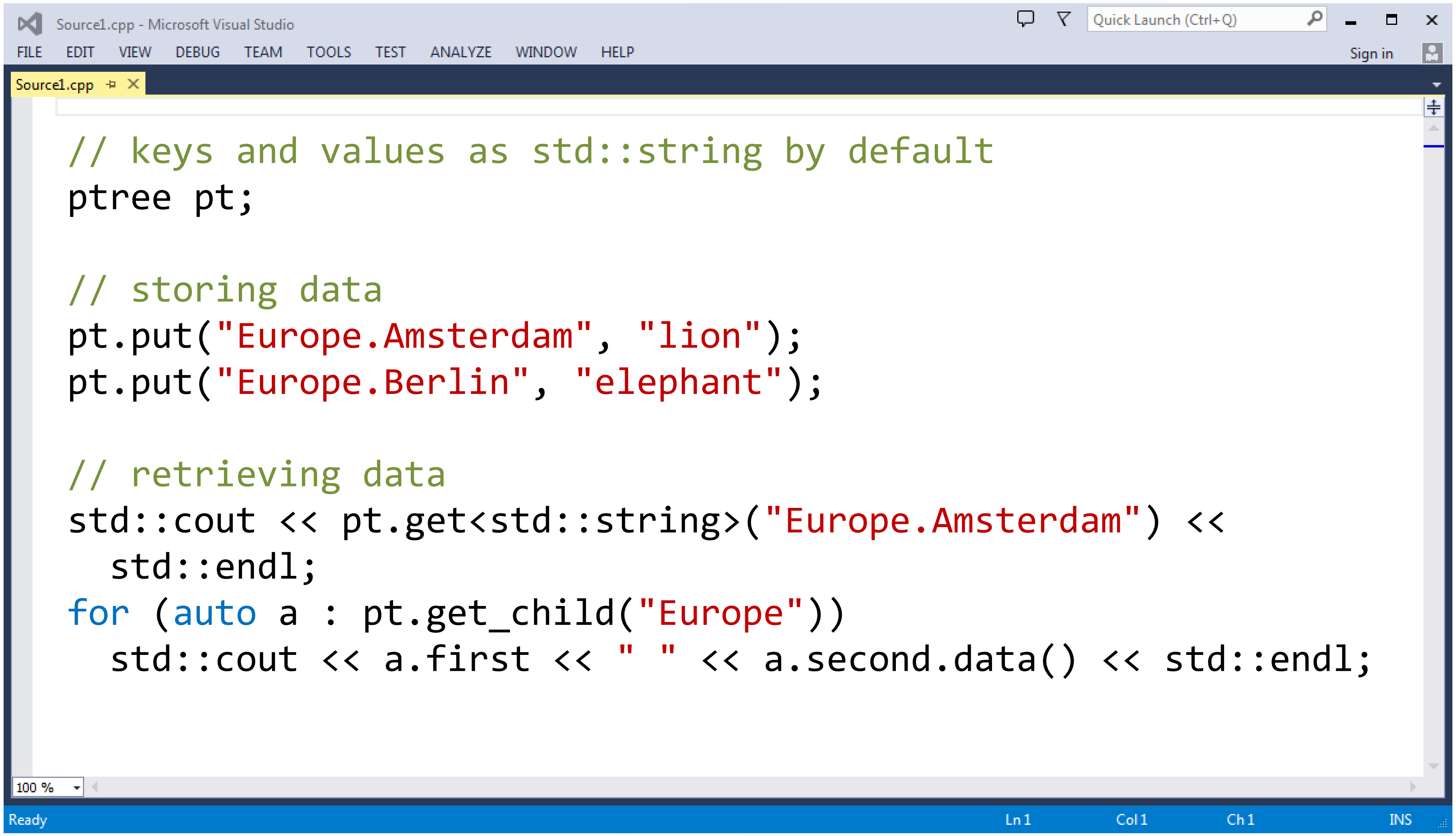
using namespace boost::property_tree;

# Boost.PropertyTree

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ☐ |
| Can be serialized with Boost.Serialization | ✓ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.41.0

# Boost.PropertyTree



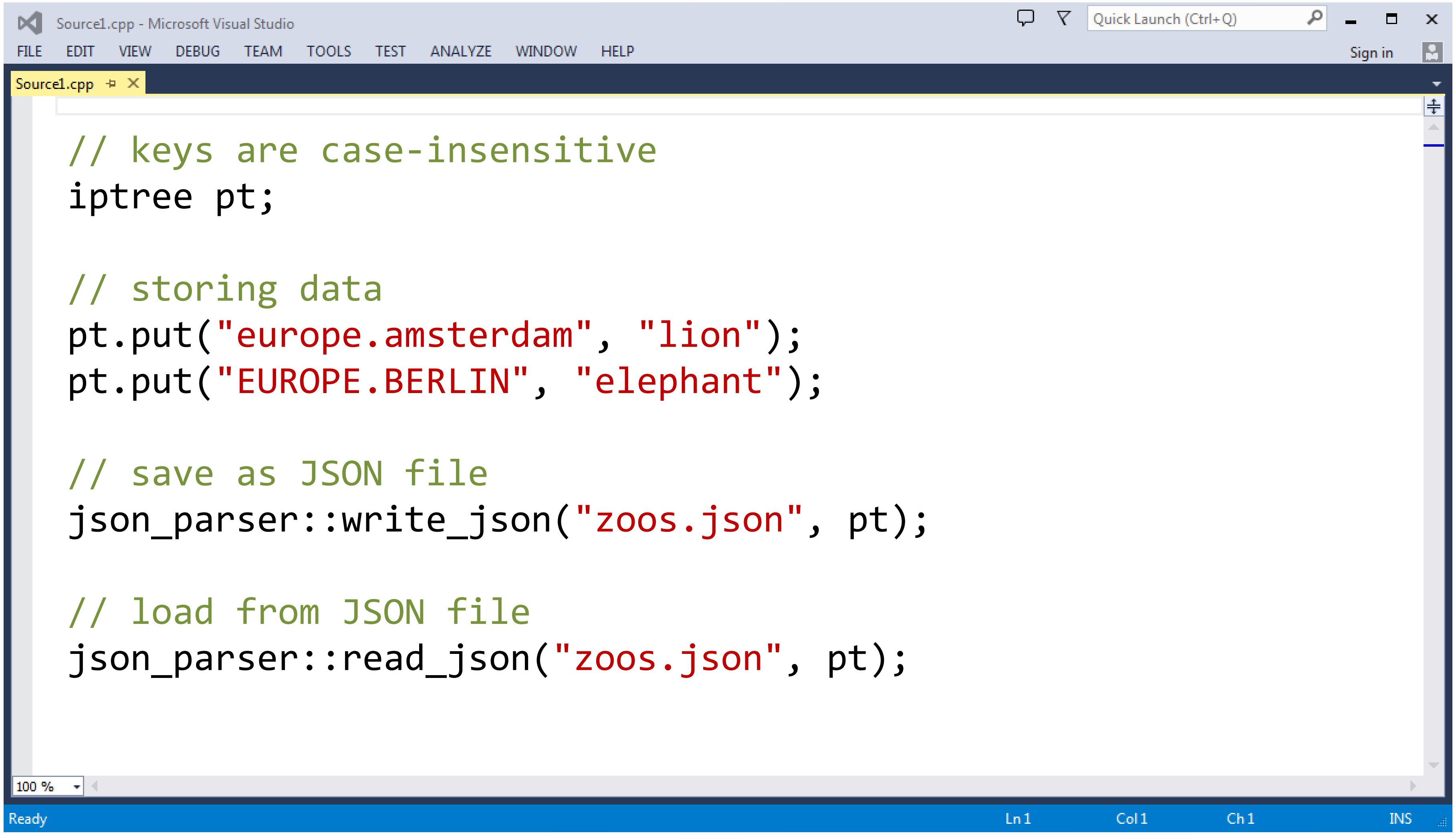
Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```
// keys and values as std::string by default
ptree pt;

// storing data
pt.put("Europe.Amsterdam", "lion");
pt.put("Europe.Berlin", "elephant");

// retrieving data
std::cout << pt.get<std::string>("Europe.Amsterdam") <<
  std::endl;
for (auto a : pt.get_child("Europe"))
  std::cout << a.first << " " << a.second.data() << std::endl;
```

# Boost.PropertyTree


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// keys are case-insensitive
iptree pt;

// storing data
pt.put("europe.amsterdam", "lion");
pt.put("EUROPE.BERLIN", "elephant");

// save as JSON file
json_parser::write_json("zoos.json", pt);

// load from JSON file
json_parser::read_json("zoos.json", pt);
```

## Boost.DynamicBitset

Works exactly like std::bitset except that the size is set (and can be changed) at runtime

- Boost has resize(), push_back() and append()
- Boost supports setting the underlying block type
- Use if you need to change size at runtime

### Header file

#include <boost/dynamic_bitset.hpp>

### Namespace

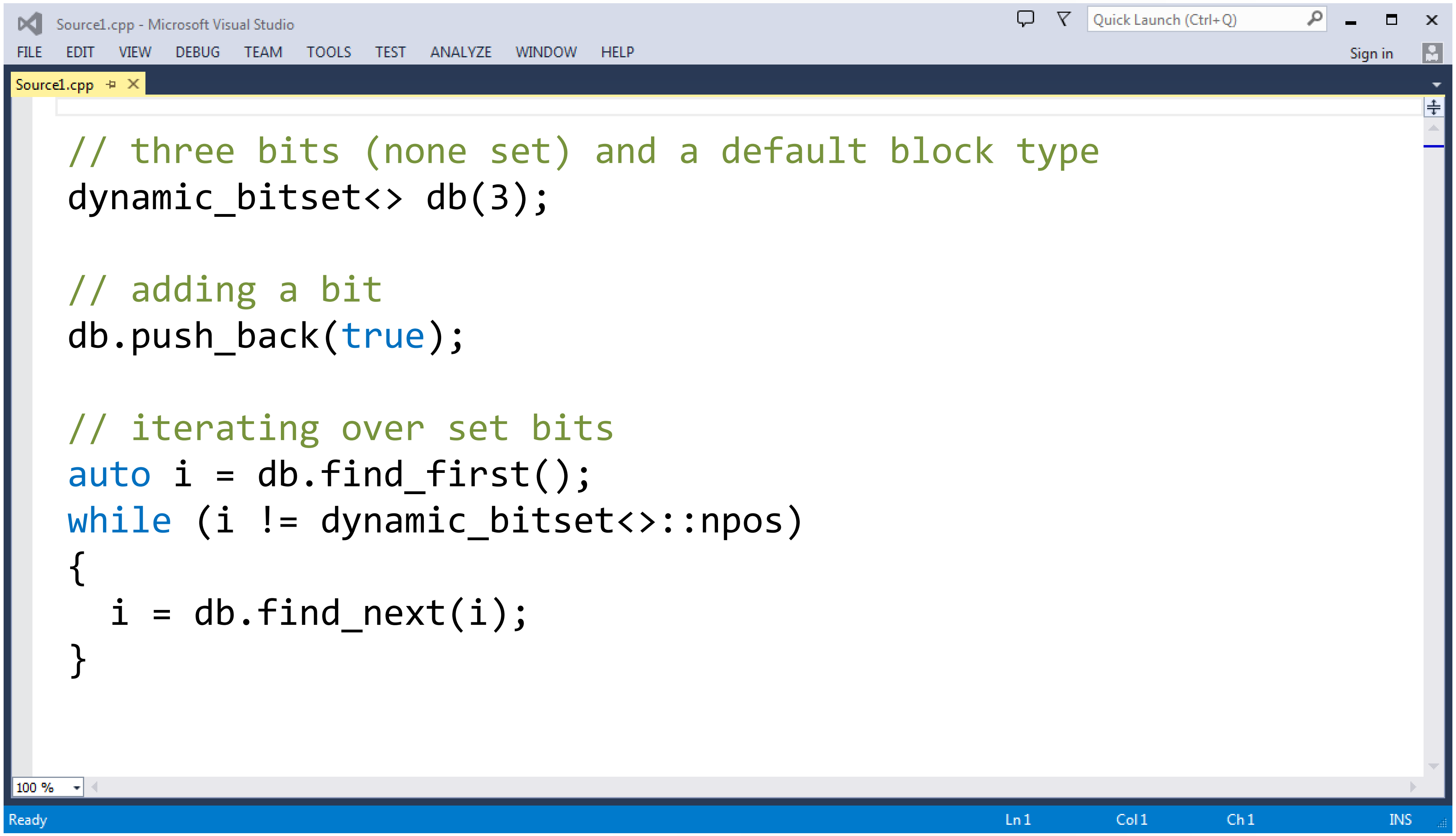
using namespace boost;

# Boost.DynamicBitset

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| ~~Validity of iterators and references preserved~~ | |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.29.0

# Boost.DynamicBitset

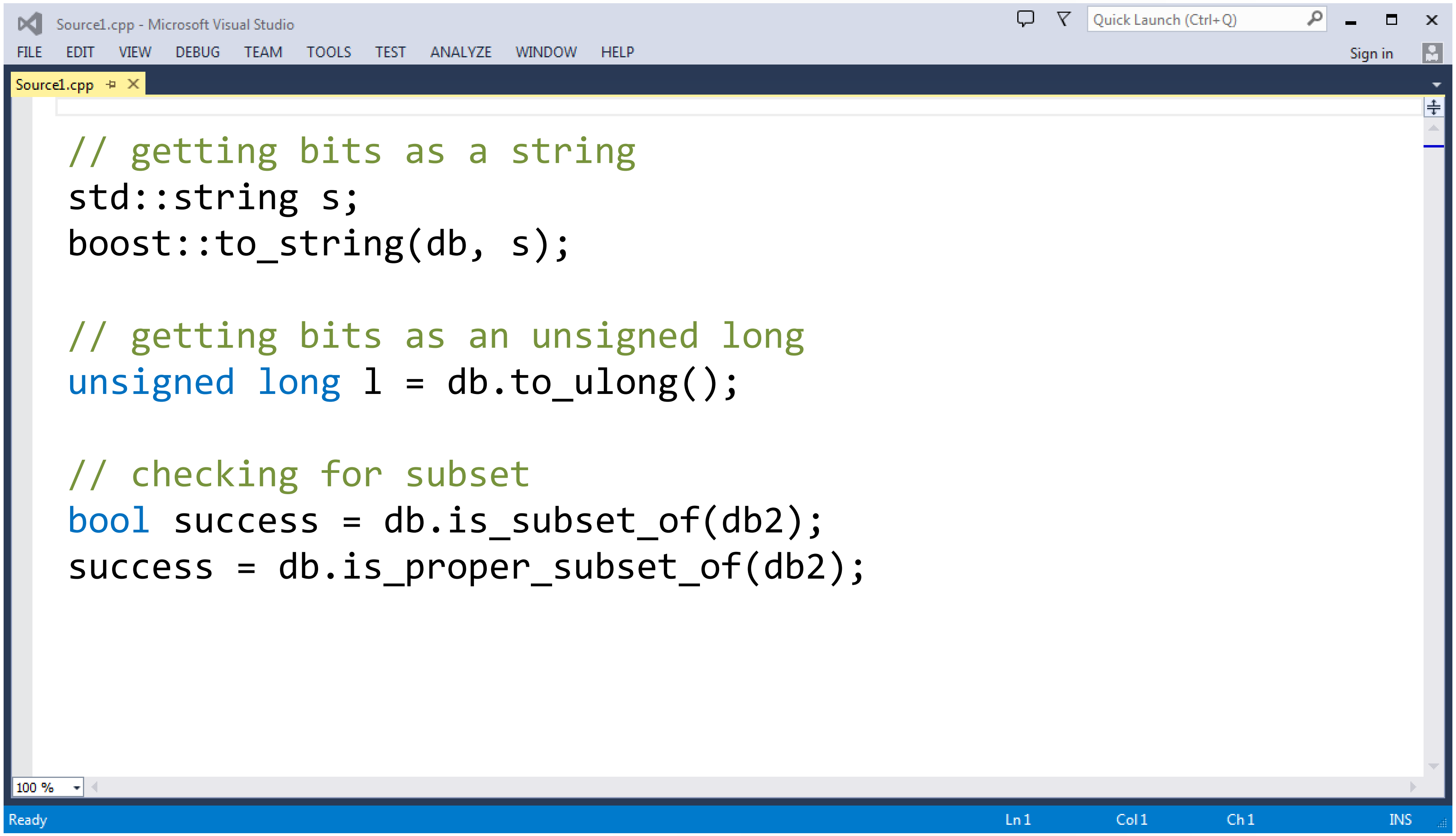

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// three bits (none set) and a default block type
dynamic_bitset<> db(3);

// adding a bit
db.push_back(true);

// iterating over set bits
auto i = db.find_first();
while (i != dynamic_bitset<>::npos)
{
  i = db.find_next(i);
}
```

# Boost.DynamicBitset


Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// getting bits as a string
std::string s;
boost::to_string(db, s);

// getting bits as an unsigned long
unsigned long l = db.to_ulong();

// checking for subset
bool success = db.is_subset_of(db2);
success = db.is_proper_subset_of(db2);
```

# Boost.Multiarray

Multi-dimensional array with number of dimensions set at compile-time and extents at runtime

- Index-based access returns a subarray
- Views to treat a part of an array as a new array
- Reshaping and resizing is supported

## Header file

#include <boost/multi_array.hpp>

## Namespace

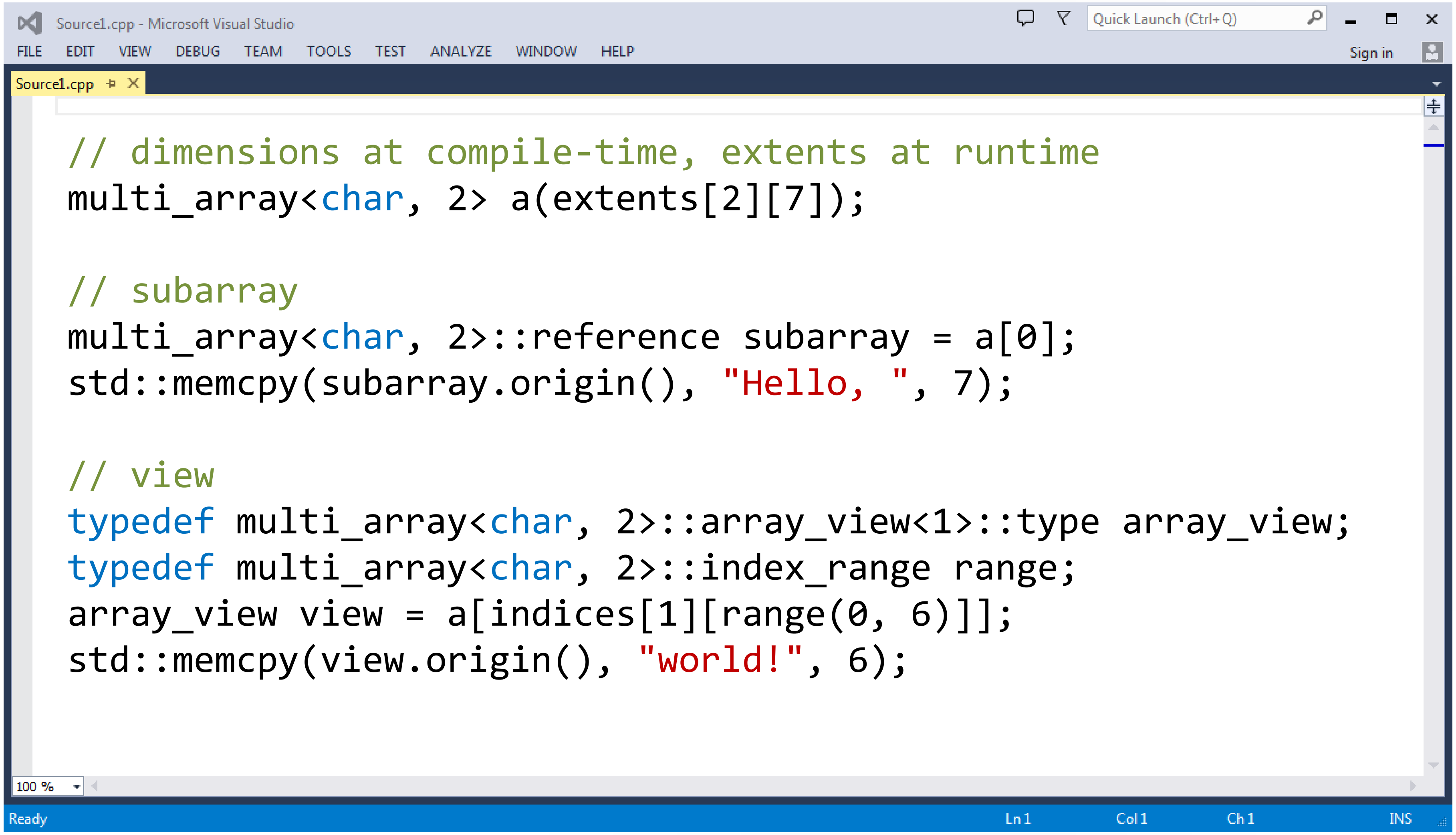
using namespace boost;

# Boost.Multiarray

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ✓ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ✓ |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.29.0

# Boost.Multiarray


Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```cpp
// dimensions at compile-time, extents at runtime
multi_array<char, 2> a(extents[2][7]);

// subarray
multi_array<char, 2>::reference subarray = a[0];
std::memcpy(subarray.origin(), "Hello, ", 7);

// view
typedef multi_array<char, 2>::array_view<1>::type array_view;
typedef multi_array<char, 2>::index_range range;
array_view view = a[indices[1][range(0, 6)]];
std::memcpy(view.origin(), "world!", 6);
```

## Boost.Heap

Priority queues like std::priority_queue but with more functionality

- Very similar interface to std::deque
- Has iterator support (random and ordered)
- Supports merging and changing elements

### Header files

#include <boost/heap/....hpp>

### Namespace

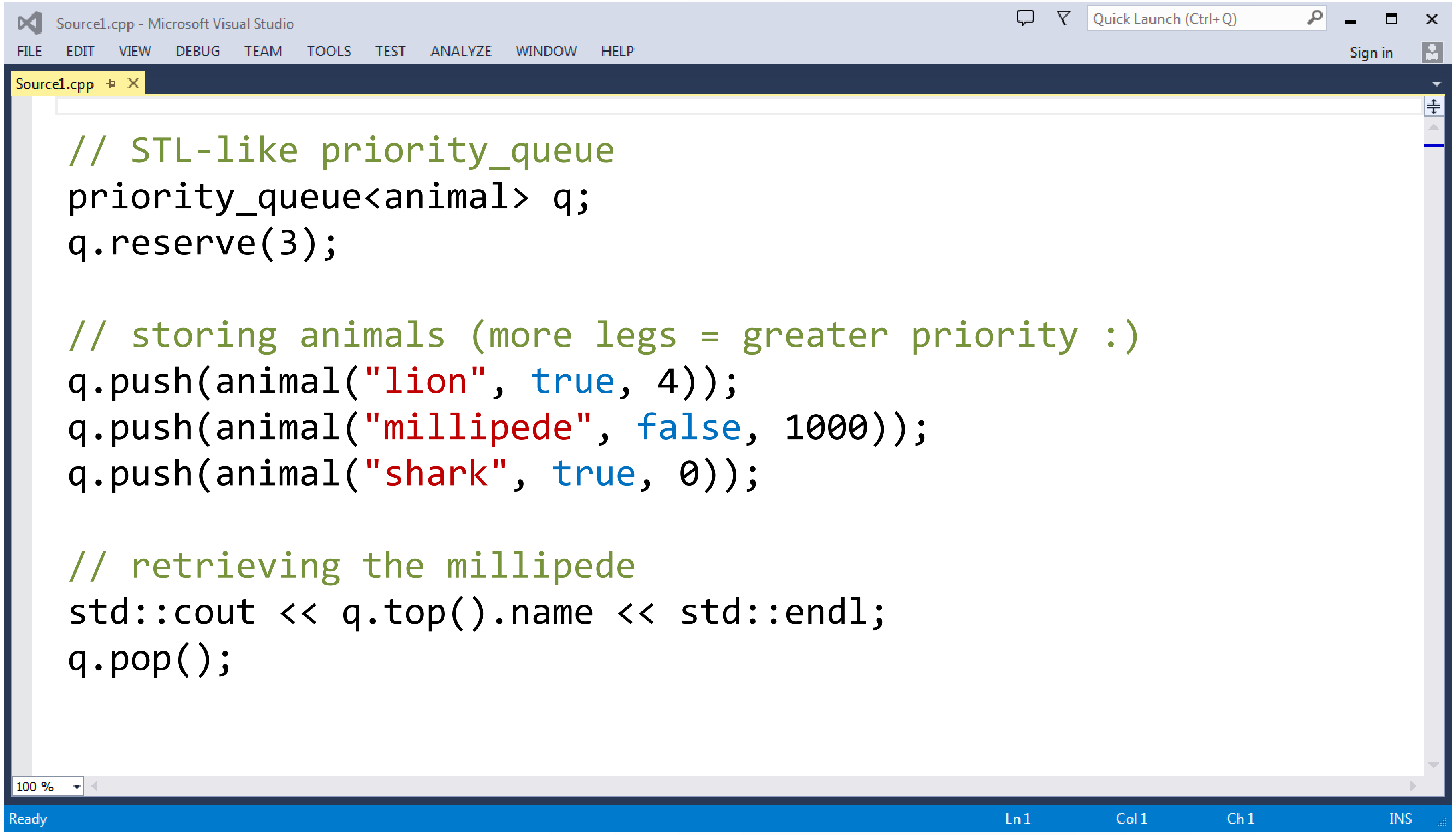
using namespace boost::heap;

# Boost.Heap

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ✓ |
| Fixed-size | ☐ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ☐ |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.49.0

# Boost.Heap

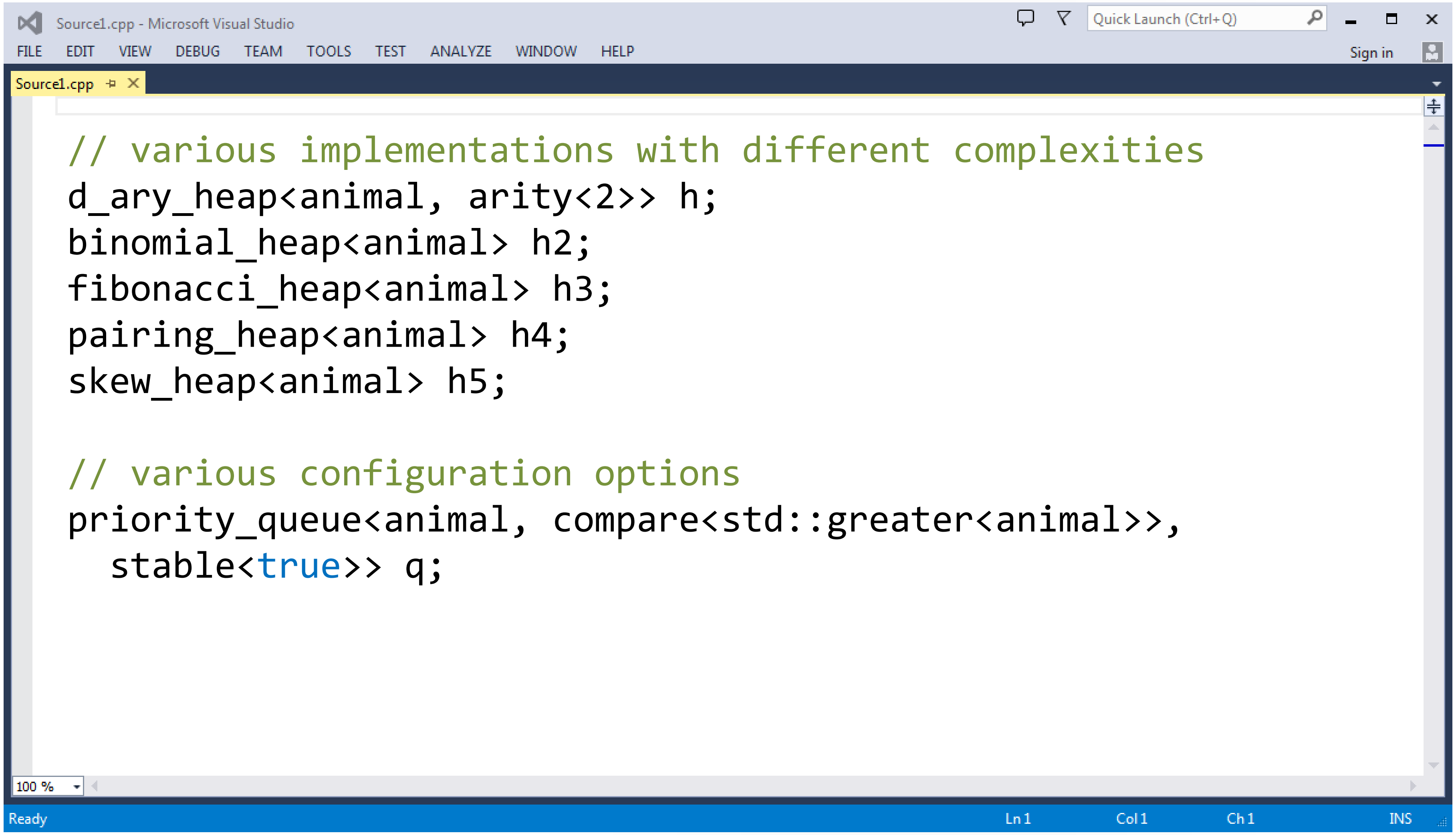

Source1.cpp - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```
// STL-like priority_queue
priority_queue<animal> q;
q.reserve(3);

// storing animals (more legs = greater priority :)
q.push(animal("lion", true, 4));
q.push(animal("millipede", false, 1000));
q.push(animal("shark", true, 0));

// retrieving the millipede
std::cout << q.top().name << std::endl;
q.pop();
```

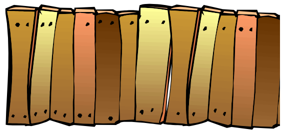
# Boost.Heap


Source1.cpp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Sign in
Source1.cpp
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```
// various implementations with different complexities
d_ary_heap<animal, arity<2>> h;
binomial_heap<animal> h2;
fibonacci_heap<animal> h3;
pairing_heap<animal> h4;
skew_heap<animal> h5;

// various configuration options
priority_queue<animal, compare<std::greater<animal>>,
  stable<true>> q;
```

# Boost.Array



A fixed-size container which looks and works like std::array from the C++ standard library

- assign() is called fill() in Boost
- C++11 has std::get<>() to fetch a value
- Just use std::array

## Header file

#include <boost/array.hpp>

## Namespace

using namespace boost;

# Boost.Array

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ☐ |
| Fixed-size | ✓ |
| Owns elements | ✓ |
| Thread-safe | ☐ |
| Validity of iterators and references preserved | ✓ |
| Can be serialized with Boost.Serialization | ☐ |
| Can be shared with Boost.Interprocess | ☐ |

Since Boost 1.17.0

## Boost.Unordered

Containers which look up elements based on hash values and look and work like the ones from the STL

- Boost uses Boost.Hash for hashing
- Just use the containers from the STL

### Header files

```
#include <boost/unordered_set.hpp>
#include <boost/unordered_map.hpp>
```

### Namespace

```
using namespace boost;
```

# Boost.Unordered

| | |
|---|---|
| C++11 support (initializer lists, move, allocators) | ✓ |
| Fixed-size | |
| Owns elements | ✓ |
| Thread-safe | |
| Validity of iterators and references preserved | ✓ |
| Can be serialized with Boost.Serialization | |
| Can be shared with Boost.Interprocess | |

Since Boost 1.36.0

# More information

- Boost documentation:

  http://www.boost.org/doc/libs

- Online book:

  http://en.highscore.de/cpp/boost/

  http://www.highscore.de/cpp/boost/ (German)

  http://zh.highscore.de/cpp/boost/ (Chinese)