CME 211: Lecture 21

Topics:

- Multi-dimensional data
- Boost `multi_array`

Layout in memory for `vector`

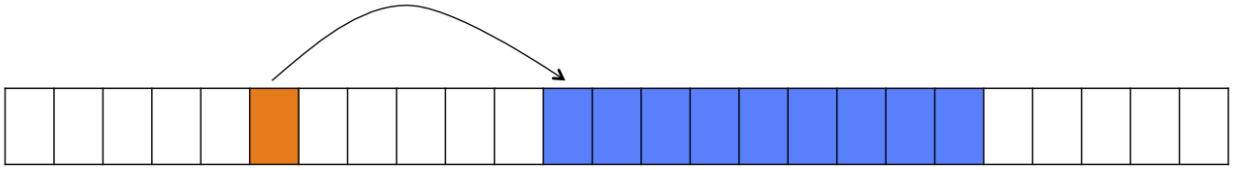

Figure 1: fig

- Memory for `std::vector` has 2 parts:
- Memory for the vector data
- Memory for the `std::vector` container. This part (essentially) includes the memory address of the vector data, the size of the vector and capacity.
- The 2 parts may be very far apart in the memory address space.

Look at the details

`src/vector_memory.cpp`:

```
#include <iostream>
#include <vector>

int main() {
  std::vector<int> a;
  for (int i = 0; i < 10; i++) {
    a.push_back(i);
  }
  std::cout << "sizeof(a): " << sizeof(a) << std::endl;
  std::cout << "    memory location of a: " << &a << std::endl;
  std::cout << " memory location of data: " << a.data() << std::endl;
  std::cout << "difference in memory loc: "
            << double((int*)&a-a.data()) / 1024 / 1024 / 1024
```

```
            << " GB" << std::endl;
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/vector_memory.cpp -o src/vector_memory
$ ./src/vector_memory
sizeof(a): 24
    memory location of a: 0x7fff57a18870
 memory location of data: 0x7ff6d3403310
difference in memory loc: 8.51711 GB
$ ./src/vector_memory
sizeof(a): 24
    memory location of a: 0x7fff56bd4870
 memory location of data: 0x7f8c5ac03310
difference in memory loc: 114.984 GB
$ ./src/vector_memory
sizeof(a): 24
    memory location of a: 0x7fff52c5e870
 memory location of data: 0x7fb08bd00040
difference in memory loc: 78.7772 GB
```

- The size of the `std::vector` container is 24 bytes, this could be for
- 8 bytes for the memory address of the vector data
- 8 bytes for the size of the vector, number of elements stored
- 8 bytes of the capacity of the vector, number of elements that may be stored before reallocation
- Memory locations are different in each run of the program. This is a security feature to make it harder to introduce malicious code or data.

Multidimensional data

- How do we handle multidimensional data in C++?

Container of containers

src/multi1.cpp:

```
#include <vector>
#include <iostream>

int main() {
  // declare vector of vectors
  std::vector< std::vector<double> > v;
  // add empty "second-level" vectors
  v.push_back(std::vector<double>());
  v.push_back(std::vector<double>());
  v.push_back(std::vector<double>());
  // add some data
  double n = 0.;
  for(unsigned int i = 0; i < 3; i++) {
    for(unsigned int j = 0; j < 3; j++) {
```

```
      v[i].push_back(n);
      n++;
    }
  }
  // print
  for(unsigned int i = 0; i < 3; i++) {
    for(unsigned int j = 0; j < 3; j++) {
      std::cout << "v[" << i << "][" << j << "] = " << v[i][j] << std::endl;
    }
  }
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/multi1.cpp -o src/multi1
$ ./src/multi1
v[0][0] = 0
v[0][1] = 1
v[0][2] = 2
v[1][0] = 3
v[1][1] = 4
v[1][2] = 5
v[2][0] = 6
v[2][1] = 7
v[2][2] = 8
```

Layout in memory

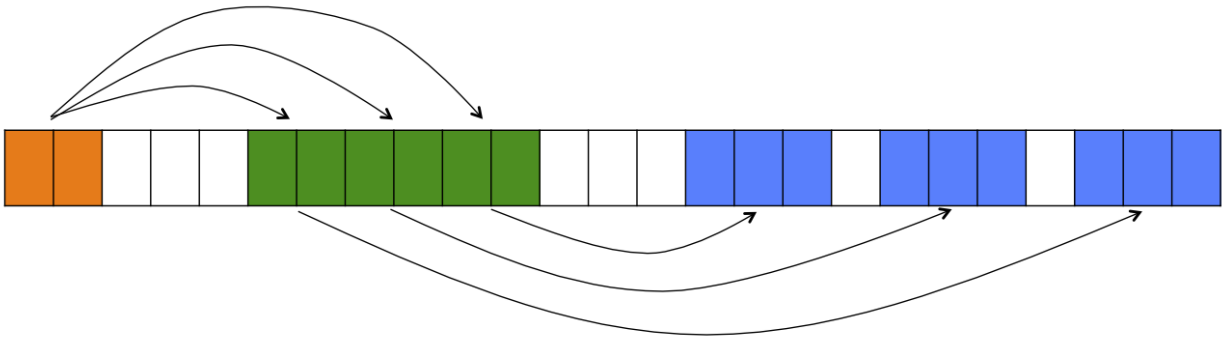

Figure 2: fig

Contiguous memory

src/multi2.cpp:

```
#include <iostream>
#include <vector>

int main() {
  unsigned int nrows = 3, ncols = 3;
  std::vector<double> a;
  a.resize(nrows*ncols);

  double n = 0.;
  for(unsigned int i = 0; i < nrows; i++) {
    for(unsigned int j = 0; j < ncols; j++) {
      // manual indexing into "multi-dimensional array"
      a[i*ncols + j] = n;
      n++;
    }
  }

  for(unsigned int i = 0; i < nrows*ncols; i++) {
    std::cout << "a[" << i << "] = " << a[i] << std::endl;
  }
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/multi2.cpp -o src/multi2
$ ./src/multi2
a[0] = 0
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
a[5] = 5
a[6] = 6
a[7] = 7
a[8] = 8
```

Boost Multidimensional Array Library

src/array1.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  unsigned int nrows = 3, ncols = 3;
  boost::multi_array<double, 2> a(boost::extents[nrows][ncols]);

  double n = 0.;
  for (unsigned int i = 0; i < nrows; i++) {
    for (unsigned int j = 0; j < ncols; j++) {
```

```
      a[i][j] = n; // access elements like static array
      n++;
    }
  }

  for (unsigned int i = 0; i < nrows; i++) {
    for (unsigned int j = 0; j < ncols; j++) {
      std::cout << "a[" << i << "][" << j << "] = " << a[i][j] << std::endl;
    }
  }
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array1.cpp -o src/array1
$ ./src/array1
a[0][0] = 0
a[0][1] = 1
a[0][2] = 2
a[1][0] = 3
a[1][1] = 4
a[1][2] = 5
a[2][0] = 6
a[2][1] = 7
a[2][2] = 8
```

Accessing the contiguous memory

src/array2.cpp:

```
#include <iostream>

#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);

  double n = 0.;
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = n;
      n++;
    }
  }

  for (unsigned int n = 0; n < a.num_elements(); n++) {
    std::cout << "a.data()[" << n << "] = " << a.data()[n] << std::endl;
  }

  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array2.cpp -o src/array2
```

```
$ ./src/array2
a.data()[0] = 0
a.data()[1] = 1
a.data()[2] = 2
a.data()[3] = 3
a.data()[4] = 4
a.data()[5] = 5
a.data()[6] = 6
a.data()[7] = 7
a.data()[8] = 8
```

Performance

src/perf1.cpp:

```
#include <iostream>
#include <ctime>
#include <boost/multi_array.hpp>

int main() {
  unsigned int nrows = 8192, ncols = 8192;
  boost::multi_array<double, 2> a(boost::extents[nrows][ncols]);

  for (unsigned int i = 0; i < nrows; i++) {
    for (unsigned int j = 0; j < ncols; j++) {
      a[i][j] = 1.0;
    }
  }

  auto t0 = std::clock();
  double sum = 0.;
  for (unsigned int i = 0; i < nrows; i++) {
    for (unsigned int j = 0; j < ncols; j++) {
      sum += a[i][j];
    }
  }
  auto t1 = std::clock();

  std::cout << " boost: sum = " << sum << ", time = "
            << double(t1-t0) / CLOCKS_PER_SEC
            << " seconds"<< std::endl;

  auto b = a.data();
  t0 = std::clock();
  sum = 0.;
  for (unsigned int n = 0; n < nrows*ncols; n++) {
    sum += b[n];
  }
  t1 = std::clock();
  std::cout << "direct: sum = " << sum << ", time = "
            << double(t1-t0) / CLOCKS_PER_SEC
            << " seconds"<< std::endl;
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/perf1.cpp -o src/perf1
$ ./src/perf1
 boost: sum = 6.71089e+07, time = 4.55984 seconds
direct: sum = 6.71089e+07, time = 0.240126 seconds
$ ./src/perf1
 boost: sum = 6.71089e+07, time = 4.41117 seconds
direct: sum = 6.71089e+07, time = 0.228782 seconds
$ ./src/perf1
 boost: sum = 6.71089e+07, time = 4.38506 seconds
direct: sum = 6.71089e+07, time = 0.235112 seconds
```

Performance

From `src/perf2.cpp`:

```
// disable boost range checking
#define BOOST_DISABLE_ASSERTS
#include <boost/multi_array.hpp>
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/perf2.cpp -o src/perf2
$ ./src/perf2
 boost: sum = 6.71089e+07, time = 4.4622 seconds
direct: sum = 6.71089e+07, time = 0.235016 seconds
$ ./src/perf2
 boost: sum = 6.71089e+07, time = 4.26261 seconds
direct: sum = 6.71089e+07, time = 0.226978 seconds
$ ./src/perf2
 boost: sum = 6.71089e+07, time = 4.2159 seconds
direct: sum = 6.71089e+07, time = 0.240271 seconds
```

Compiler optimization

Enable compiler optimizations with the `-O3` argument.

With range checking:

Output:

```
$ clang++ -O3 -std=c++11 -Wall -Wextra -Wconversion src/perf1.cpp -o src/perf1
$ ./src/perf1
 boost: sum = 6.71089e+07, time = 0.102904 seconds
direct: sum = 6.71089e+07, time = 0.107179 seconds
$ ./src/perf1
 boost: sum = 6.71089e+07, time = 0.119958 seconds
direct: sum = 6.71089e+07, time = 0.121643 seconds
$ ./src/perf1
 boost: sum = 6.71089e+07, time = 0.10259 seconds
direct: sum = 6.71089e+07, time = 0.105372 seconds
```

Range checking disabled:

Output:

```
$ clang++ -O3 -std=c++11 -Wall -Wextra -Wconversion src/perf2.cpp -o src/perf2
$ ./src/perf2
 boost: sum = 6.71089e+07, time = 0.102209 seconds
direct: sum = 6.71089e+07, time = 0.102016 seconds
$ ./src/perf2
 boost: sum = 6.71089e+07, time = 0.104234 seconds
direct: sum = 6.71089e+07, time = 0.105256 seconds
$ ./src/perf2
 boost: sum = 6.71089e+07, time = 0.101876 seconds
direct: sum = 6.71089e+07, time = 0.117592 seconds
```

Range checking

src/array3a.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);
  a[3][3] = 1.;
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array3a.cpp -o src/array3a
$ ./src/array3a
Assertion failed: (size_type(idx - index_bases[0]) < extents[0]), function access, file /usr/local/inclu
```

Range checking

src/array3b.cpp:

```
#include <iostream>
#define BOOST_DISABLE_ASSERTS
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);
  a[3][3] = 1.;
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array3b.cpp -o src/array3b
$ ./src/array3b
$ clang++ -std=c++11 -g -fsanitize=address -Wall -Wextra -Wconversion src/array3b.cpp -o src/array3b
$ ./src/array3b
=================================================================
==36808==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60700000df30 at pc 0x00010dc1aad5 bp
WRITE of size 8 at 0x60700000df30 thread T0
    #0 0x10dc1aad4 in main array3b.cpp:7
    #1 0x7fff9426d5ac in start (libdyld.dylib+0x35ac)
    #2 0x0  (<unknown module>)
```

```
0x60700000df30 is located 16 bytes to the left of 67-byte region [0x60700000df40,0x60700000df83)
allocated by thread T0 here:
    #0 0x10dc759c0 in wrap_malloc (libclang_rt.asan_osx_dynamic.dylib+0x489c0)
    #1 0x7fff9283364d in _xpc_malloc (libxpc.dylib+0x264d)
    #2 0x7fff92833529 in _xpc_dictionary_insert (libxpc.dylib+0x2529)
    #3 0x7fff92833325 in xpc_dictionary_set_string (libxpc.dylib+0x2325)
    #4 0x7fff9283319c in _xpc_collect_environment (libxpc.dylib+0x219c)
    #5 0x7fff92832d41 in _libxpc_initializer (libxpc.dylib+0x1d41)
    #6 0x7fff924bfa06 in libSystem_initializer (libSystem.B.dylib+0x1a06)
    #7 0x7fff6536e10a  (<unknown module>)
    #8 0x7fff6536e283  (<unknown module>)
    #9 0x7fff6536a8bc  (<unknown module>)
    #10 0x7fff6536a851  (<unknown module>)
    #11 0x7fff6536a851  (<unknown module>)
    #12 0x7fff6536a851  (<unknown module>)
    #13 0x7fff6536a742  (<unknown module>)
    #14 0x7fff6536a9b2  (<unknown module>)
    #15 0x7fff6535d0f0  (<unknown module>)
    #16 0x7fff65360d97  (<unknown module>)
    #17 0x7fff6535c275  (<unknown module>)
    #18 0x7fff6535c035  (<unknown module>)
    #19 0x0  (<unknown module>)

SUMMARY: AddressSanitizer: heap-buffer-overflow array3b.cpp:7 in main
Shadow bytes around the buggy address:
  0x1c0e00001b90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001ba0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001bb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001bc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001bd0: fa fa fa fa fa fa fa fa fa fa 00 00 00 00 00 00
=>0x1c0e00001be0: 00 00 00 fa fa fa[fa]fa 00 00 00 00 00 00 00 00
  0x1c0e00001bf0: 03 fa fa fa fa fa 00 00 00 00 00 00 00 00 00 fa
  0x1c0e00001c00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001c10: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001c20: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0e00001c30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
```

```
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==36808==ABORTING
```

Range checking

Another method to check for memory leaks is `valgrind`.

Output:

```
$ clang++ -g -Wall -Wextra -Wconversion src/array3b.cpp -o src/array3b
$ valgrind ./src/array3b
==36817== Memcheck, a memory error detector
==36817== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==36817== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==36817== Command: ./src/array3b
==36817==
==36817== Invalid write of size 8
==36817==    at 0x10000109D: main (array3b.cpp:7)
==36817==  Address 0x100a88180 is 16 bytes after a block of size 80 in arena "client"
==36817==
==36817==
==36817== HEAP SUMMARY:
==36817==     in use at exit: 22,458 bytes in 194 blocks
==36817==   total heap usage: 258 allocs, 64 frees, 28,226 bytes allocated
==36817==
==36817== LEAK SUMMARY:
==36817==    definitely lost: 0 bytes in 0 blocks
==36817==    indirectly lost: 0 bytes in 0 blocks
==36817==      possibly lost: 2,064 bytes in 1 blocks
==36817==    still reachable: 0 bytes in 0 blocks
==36817==         suppressed: 20,394 bytes in 193 blocks
==36817== Rerun with --leak-check=full to see details of leaked memory
==36817==
==36817== For counts of detected and suppressed errors, rerun with: -v
==36817== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Elementwise comparison

src/array5.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);
  boost::multi_array<double, 2> b(boost::extents[3][3]);

  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = 1.;
      b[i][j] = 2.;
```

```
    }
  }

  std::cout << "a == b: " << (a == b) << std::endl;
  std::cout << "a < b: " << (a < b) << std::endl;
  std::cout << "a > b: " << (a > b) << std::endl;

  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array5.cpp -o src/array5
$ ./src/array5
a == b: 0
a < b: 1
a > b: 0
```

Copy or reference?

src/array6a.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);

  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = 1.;
    }
  }

  auto b = a; // copy or reference?

  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = 2.;
    }
  }

  std::cout << "a b" << std::endl;
  std::cout << "---" << std::endl;
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      std::cout << a[i][j] << " " << b[i][j] << std::endl;
    }
  }
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array6a.cpp -o src/array6a
$ ./src/array6a
```

```
a b
---
2 1
2 1
2 1
2 1
2 1
2 1
2 1
2 1
2 1
```

Passing an array to a function

src/array6b.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

void increment(boost::multi_array<double, 2> b) {
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      b[i][j]++;
    }
  }
}

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);

  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = 1.;
    }
  }

  increment(a);

  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      std::cout << a[i][j] << std::endl;
    }
  }
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array6b.cpp -o src/array6b
$ ./src/array6b
1
1
1
1
1
```

```
1
1
1
1
```

Passing by reference

From `src/array6c.cpp`:

```
void increment(boost::multi_array<double, 2>& b) {
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      b[i][j]++;
    }
  }
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array6c.cpp -o src/array6c
$ ./src/array6c
2
2
2
2
2
2
2
2
2
```

Array operations?

- Boost `multi_array` does not support array operations like NumPy
- If `a` is a `multi_array` things like `2*a` and `a = 1.0` will not work and will lead to very long compiler error messages.
- If you want this kind of stuff, have a look at:
- http://eigen.tuxfamily.org/index.php?title=Main_Page
- http://arma.sourceforge.net/

Array views

An **array view** is essentially a reference into a sub-array of a larger array.

`src/array9.cpp`:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);
```

```
  double n = 0.;
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = n;
      n++;
    }
  }

  /* Setup b as a view into a subset of a. */
  typedef boost::multi_array<double, 2>::index_range index_range;
  auto b = a[boost::indices[index_range(1,3)][index_range(1,3)]];

  for (unsigned int i = 0; i < 2; i++) {
    for (unsigned int j = 0; j < 2; j++) {
      b[i][j] = -1.;
    }
  }

  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      std::cout << a[i][j] << std::endl;
    }
  }
  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array9.cpp -o src/array9
$ ./src/array9
0
1
2
3
-1
-1
6
-1
-1
```

Storage order

src/array10a.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3]);

  double n = 0.;
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = n;
      n++;
```

```
    }
  }

  auto b = a.data();
  for (unsigned int n = 0; n < a.num_elements(); n++) {
    std::cout << "b[" << n << "] = " << b[n] << std::endl;
  }

  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array10a.cpp -o src/array10a
$ ./src/array10a
b[0] = 0
b[1] = 1
b[2] = 2
b[3] = 3
b[4] = 4
b[5] = 5
b[6] = 6
b[7] = 7
b[8] = 8
```

- Uses C convention that rows are stored contiguously in memory (row major order)
- Or put another way, the last index in a multidimensional array changes fastest when traversing through linear memory

"Fortran" storage order

src/array10b.cpp:

```
#include <iostream>
#include <boost/multi_array.hpp>

int main() {
  boost::multi_array<double, 2> a(boost::extents[3][3],
                                  boost::fortran_storage_order());

  double n = 0.;
  for (unsigned int i = 0; i < 3; i++) {
    for (unsigned int j = 0; j < 3; j++) {
      a[i][j] = n;
      n++;
    }
  }

  auto b = a.data();
  for (unsigned int n = 0; n < a.num_elements(); n++) {
    std::cout << "b[" << n << "] = " << b[n] << std::endl;
  }

  return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array10b.cpp -o src/array10b
$ ./src/array10b
b[0] = 0
b[1] = 3
b[2] = 6
b[3] = 1
b[4] = 4
b[5] = 7
b[6] = 2
b[7] = 5
b[8] = 8
```

- In Fortran columns are stored contiguously in memory (column major order)
- Or put another way, the first index in a multidimensional array changes fastest when traversing through linear memory

MultiArrays are containers

From `src/accumulate.cpp`:

```
  for (unsigned int i = 0; i < nrows; i++) {
    sum += std::accumulate(a[i].begin(), a[i].end(), 0.);
  }
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/accumulate.cpp -o src/accumulate
$ ./src/accumulate
 boost: sum = 6.71089e+07, time = 4.49544 seconds
direct: sum = 6.71089e+07, time = 0.235229 seconds
 accum: sum = 6.71089e+07, time = 3.02097 seconds
```

Boost summary

From http://www.boost.org:

Boost provides free peer-reviewed portable C++ source libraries.

We emphasize libraries that work well with the C++ Standard Library. Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications. The Boost license encourages both commercial and non-commercial use.

Good:

- Well implemented library with a lot of diverse functionality.
- Approximately 115 sub-libraries, of which MultiArray is just one of them.
- Cross platform (Windows, Mac, Linux) and friendly license for commercial applications.

Bad:

- Sometimes the documentation can be a bit lacking.
- Not a standard part of C++ (external dependency).
- Some people seem to have a real aversion to it.

- Sometimes the `boost` library authors make an effort to utilize C++ features at the expense of code clarity. I believe this is why some people have strong feelings against `boost`.

Practical advice:

- Use boost if it helps you get your work done quickly.
- If you find yourself trying too hard to fit into a particular boost library, then maybe look for something else.
- It is sometimes nice to have single external dependency that contains many useful utilities as opposed to many smaller external dependencies.