Using the New C++11 Array and Tuple Containers

by Darryl Gove and Steve Clamage

The C++11 standard introduces a couple of very useful container types: arrays and tuples.

Published July 2014

About Arrays

The C++ 11 standard introduces a `std::array` which is equivalent to a traditional fixed length array, but is accessible through standard container methods. An array is potentially faster than a vector since it is a fixed size. Since the size of the array is known, an implementation can place the data on the stack with a locally declared `std::array` object.

Listing 1 shows how an array might be used.

```
#include <array>
#include <algorithm>
#include <iostream>

int main()
{
  const int arraysize = 10;
  std::array <int, arraysize> a;
  for (int i=0; i<arraysize; i++)
  {
    a[i]=(( i*32315^393) &15);
  }
  std::for_each( a.begin(), a.end(),
                [](int v ) { std::cout << v << '\n'; } );
}
```

Listing 1. Using an array

The output from compiling and running this code is shown in Listing 2.

```
$ CC -O -std=c++11 array.cpp
bash-3.2$ ./a.out
9
2
15
8
5
14
11
4
1
10
```

Listing 2. Compiling and running array example

The `std::for_each` loop in Listing 1 can also be rewritten using a range-based `for`, as shown in Listing 3. This is a shorter way of writing the same code.

```
#include <array>
#include <iostream>

int main()
{
    std::array <int,10> a;
    for (int i=0; i<10; i++)
      { a[i]=(( i*32315^393) &15); }

    for( int v : a ) std::cout << v << '\n';
}
```

Listing 3. Using range-based `for`

About Tuples

A tuple is a generalization of a pair, an ordered set of heterogeneous elements. One way to imagine using a tuple is to hold a row of data in a database. The row might contain the attributes of a person, such as the person's name, age, height, and so on. All the elements might have different types, but they all belong together as one row.

In Listing 4 we store multiple tuples of integer values into a vector, and then print them out.

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

typedef std::tuple<int,int,int> i3tuple;

int main()
{
  std::vector <i3tuple> v;
  for (int i=0; i<10; i++)
```

About the Authors

Darryl Gove is a senior principal software engineer in the Oracle Solaris Studio team, working on optimizing applications and benchmarks for current and future processors. He is also the author of the books Multicore Application Programming, Solaris Application Programming, and The Developer's Edge. Find his blog at http://blogs.oracle.com/d.

Steve Clamage is a senior engineer in the Oracle Solaris Studio team and project lead for Oracle's C++ compiler. He has been the chair of the U.S. C++ Standards Committee since 1996.

See Also

- RAW pipeline hazards
- Oracle Solaris Studio
- SPARC Systems
- Studio Forums
- Tech Articles for Developers

Follow OTN

OTN Homepage

```
  {
    v.push_back(i3tuple(i,i*2, i*2+1) );
  }

  for(i3tuple t: v)
  {
    std::cout << std::get<0>(t) << ' ';
    std::cout << std::get<1>(t) << ' ';
    std::cout << std::get<2>(t) << '\n';
  };
}
```

Listing 4. Storing and retrieving tuples from a vector

The results of compiling and running this code are shown in Listing 5.

```
$ CC -O -std=c++11 tuple.cpp
bash-3.2$ ./a.out
0 0 1
1 2 3
2 4 5
3 6 7
4 8 9
5 10 11
6 12 13
7 14 15
8 16 17
9 18 19
```

Listing 5. Compiling and running the tuple example

It is possible to create tuples without explicitly declaring the types, as shown in Listing 6.

```
#include <tuple>
#include <iostream>

int main()
{
  auto t = std::make_tuple("String",5.2, 1);
  std::cout << std::get<0>(t) << ' '
            << std::get<1>(t) << ' '
            << std::get<2>(t) << '\n';
}
```

Listing 6. Implicitly declaring a tuple

The output from the code in Listing 6 is shown in Listing 7.

```
$ CC -O -std=c++11 implicit_tuple.cpp
$ ./a.out
String 5.2 1
```

Listing 7. Output using implicitly declared tuple

Conclusion

The new `std::array` and `std::tuple` containers provide developers with additional ways to manage structured data efficiently.

Revision 1.1, 08/28/2014

Follow us:
Blog | Facebook | Twitter | YouTube