STL Algorithms - Principles and Practice

Victor Ciura - Technical Lead, Advanced Installer

Gabriel Diaconița - Senior Software Developer, Advanced Installer

http://www.advancedinstaller.com

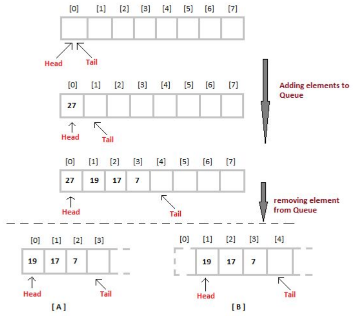
CAPHYON

December 2016

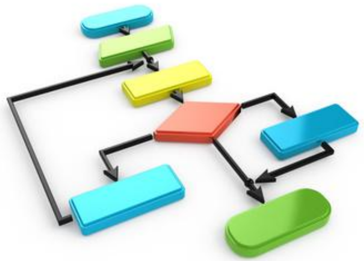
Agenda

Part 0: STL Background

Part 1: Containers and Iterators



Part 2-3: STL Algorithms Principles and Practice



Part 4: STL Function Objects and Utilities

STL Algorithms - Principles and Practice

(Part 2)

“Show me the code”

Extend STL With Your Generic Algorithms

Eg.

```
template<class Container, class Value>
void name_this_algorithm(Container & c, const Value & v)
{
  if ( find(begin(c), end(c), v) == end(c) )
    c.emplace_back(v);

  assert( !c.empty() );
}
```

Extend STL With Your Generic Algorithms

Eg.

```
template<class Container, class Value>
bool erase_if_exists(Container & c,
                     const Value & v)
{
  auto found = std::find(begin(c), end(c), v);
  if (found != end(v))
  {
    c.erase(found); // call 'erase' from STL container
    return true;
  }
  return false;
}
```

Consider Adding Range-based Versions of STL Algorithms

```
namespace range {   // our <algorithm_range.h> has ~150 wrappers for std algorithms

  template< class InputRange, class T > inline
  typename auto find(InputRange && range, const T & value)
  {
    return std::find(begin(range), end(range), value);
  }

  template< class InputRange, class UnaryPredicate > inline
  typename auto find_if(InputRange && range, UnaryPredicate pred)
  {
    return std::find_if(begin(range), end(range), pred);
  }

  template< class RandomAccessRange, class BinaryPredicate > inline
  void sort(RandomAccessRange && range, BinaryPredicate comp)
  {
    std::sort(begin(range), end(range), comp);
  }

}
```

Consider Adding Range-based Versions of STL Algorithms

Eg.

```
vector<string> v = { … };

auto it = range::find(v, "stl");
string str = *it;

auto chIt = range::find(str, 't');

auto it2 = range::find_if(v, [](const auto & val) { return val.size() > 5; });

range::sort(v);

range::sort(v, [](const auto & val1, const auto & val2)
               { return val1.size() < val2.size(); } );
```

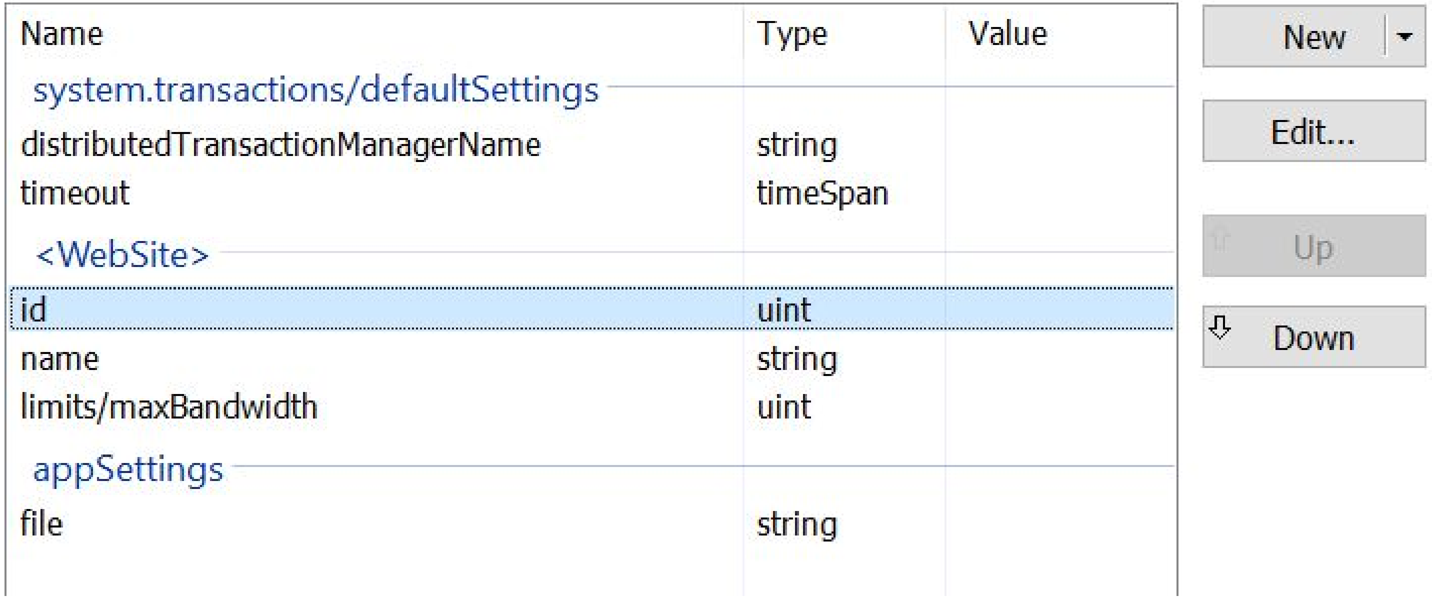
Calculating total number of unread messages.

```
// Raw loop version.  See anything wrong?
int MessagePool::CountUnreadMessages()  const
{
  int unreadCount = 0;

  for (size_t i = 0; i < mReaders.size(); ++i)
  {
      const vector<MessageItem *> & readMessages = Readers[i]->GetMessages();

      for (size_t j = 0; j < readMessages.size(); ++i)
      {
        if ( ! readMessages[j]->mRead )
          unreadCount++;
      }
  }
  return unreadCount;
}
```

Our own code. Calculating total number of unread messages.

```
// Modern C++, with STL:
int MessagePool::CountUnreadMessages()  const
{
  return std::accumulate(
    begin(mReaders), end(mReaders),  0,
    [](int count, auto & reader)
    {
      const auto & readMessages = reader->GetMessages();

      return count + std::count_if( begin(readMessages),
                                    end(readMessages),
                                    []( const auto & message)
                                    {
                                       return ! message->mRead;
                                    });
    });
}
```

Our own code. Enabling move operation (up/down) for a List item in user interface



Name	Type	Value
system.transactions/defaultSettings		
distributedTransactionManagerName	string	
timeout	timeSpan	
<WebSite>		
id	uint	
name	string	
limits/maxBandwidth	uint	
appSettings		
file	string	

Our own code. Enabling move operation (up/down) for a List item in user interface

```
// Modern version, STL algorithm based
bool CanListItemBeMoved(ListRow & aCurrentRow,  bool aMoveUp) const
{
  vector<ListRow *> existingRows = GetListRows( aCurrentRow.GetGroup() );

  auto minmax = std::minmax_element(begin(existingRows),
                                    end(existingRows),
                                    []( auto & firstRow, auto & secondRow)
                                    {
                                      return firstRow.GetOrderNumber() <
                                             secondRow.GetOrderNumber();
                                    });

  if (aMoveUp)
    return (*minmax.first)->GetOrderNumber() < aCurrentRow.GetOrderNumber();
  else
    return (*minmax.second)->GetOrderNumber() > aCurrentRow.GetOrderNumber();
}
```

Enabling move operation (up/down) for a List item in user interface

```
// Raw loop version, See anything wrong?
bool CanListItemBeMoved(ListRow & aCurrentRow, bool aMoveUp)  const
{
  int min, max;
  vector<ListRow *> existingProperties = GetListRows(aCurrentRow.GetGroup());

  for (int i = 0; i < existingProperties.size(); ++i)
  {
      const int currentOrderNumber = existingProperties[i]->GetOrderNumber();
      if (currentOrderNumber < min)
          min = currentOrderNumber;
      if (currentOrderNumber > max)
          max = currentOrderNumber;
  }
  if (aMoveUp)
    return min < aCurrentRow.GetOrderNumber();
  else
    return max > aCurrentRow.GetOrderNumber();
}
```

Our own code. Selecting attributes from XML nodes.

```
vector<XmlDomNode> childrenVector = parentNode.GetChildren(childrenVector);

set<wstring> childrenNames;
std::transform(begin(childrenVector), end(childrenVector),
               inserter(childrenNames, begin(childrenNames)),
                        getNodeNameLambda);


// A good, range based for, alternative:

for (auto & childNode : childrenVector)
    childrenNames.insert(getNodeNameLambda(childNode));


// Raw loo , see anything wrong?

for (unsigned int i = childrenVector.size(); i >= 0; i -= 1)
  childrenNames.insert(getNodeNameLambda(childrenVector[i]));
```

Demo: Server Nodes

We have a huge network of server nodes.

Each server node contains a copy of a particular ***data*** `Value` (not necessarily unique).

`class Value` is a ***Regular*** type.

{ *Assignable* + *Constructible* + *EqualityComparable* + *LessThanComparable* }

The network is constructed in such a way that the nodes are ***sorted ascending*** with respect to their `Value` but their sequence might be **rotated** (left) by some offset.

Eg.

For the ***ordered*** node values:

```
{ A, B, C, D, E, F, G, H }
```

The actual network configuration might look like:

```
{ D, E, F, G, H, A, B, C }
```

Demo: Server Nodes

The network exposes the following APIs:

```
// gives the total number of nodes - O(1)
size_t Count() const;

// retrieves the data from a given node - O(1)
const Value & GetData(size_t index) const;

// iterator interface for the network nodes
vector<Value>::const_iterator BeginNodes() const;
vector<Value>::const_iterator EndNodes() const;
```

Implement a new API for the network, that efficiently finds a server node (address) containing a given data `Value`.

```
size_t GetNode(const Value & data) const;
```

Demo: Server Nodes

```
// Code walk-through
```

Student solutions for Homework

Homework 1 : `IterateSecond()` **adapter**

- Cioarec Alexandru Marian

Homework 2 : STL Snake game

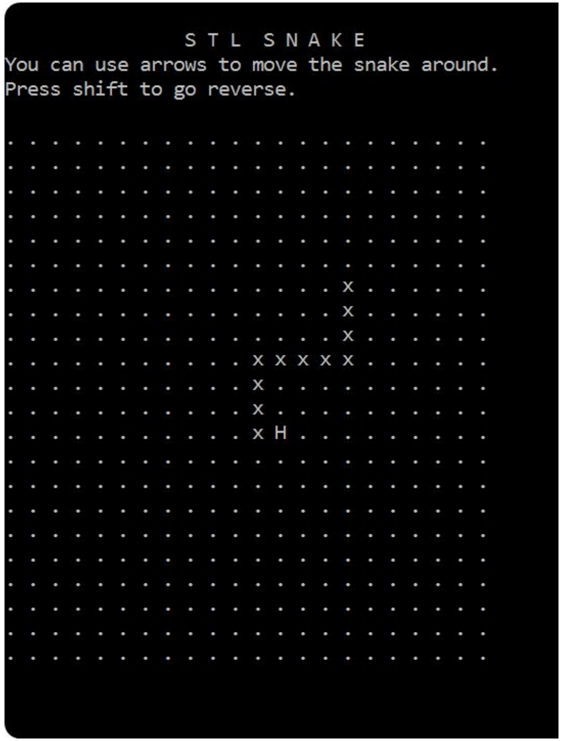
- Cioabla Alin
- Mihai Cătălin

Partial solutions:

- Cioarec Alexandru Marian
- Alexandru Chiurtu
- Mihai Predoaica
- Ciurcea Daniel

Time for coding fun!

We have a little game for you to refactor, using **STL**



```
                S T L  S N A K E
You can use arrows to move the snake around.
Press shift to go reverse.

. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . x . . . . . .
. . . . . . . . . . . . . . . x . . . . . .
. . . . . . . . . . . . . . . x . . . . . .
. . . . . . . . . . . x x x x x . . . . . .
. . . . . . . . . . . x . . . . . . . . . .
. . . . . . . . . . . x . . . . . . . . . .
. . . . . . . . . . . x H . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . .
```

Open with Visual Studio 2015

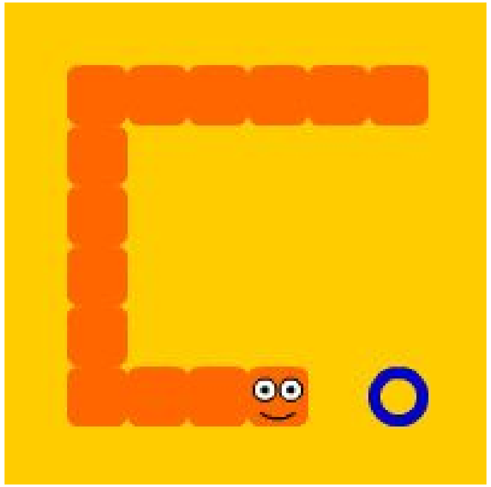
Search for **#STL** blocks

Refactor C-style **#STL** blocks using valid STL code

Is the snake still snakin’ & dyin’ right?

Demo: STL Snake

```
// Code walk-through
```



STL for Competitive Programming and Software Development
Coding Test - April 2016

You are given a keywords “database” in the form of a large text file (**`keywords.db`** ~ 136MB) containing search terms (phrases) used by people in the past (consider this an active search cache). Here is a small *fragment* from this text file:

```
---------------------- keywords.db ---------------------------
philips lcd 15
15 lcd cheap monitor
cheap 15 lcd monitor
dell e153fp 15 lcd midnight grey 36
lcd tv 15
samsung lcd 15
sony 15 lcd monitor
15 dvd lcd tv
15 inch lcd plasma monitors
--------------------------------------------------------------------
```

Suggested searches for keyword: **`"cruise"`**

```
> cruise line
> cruise ship
> carnival cruise
> caribbean cruise
> princess cruise
> disney cruise
> celebrity cruise
> norwegian cruise
```

STL for Competitive Programming and Software Development
Coding Test - April 2016

➽ Your first task is to load and **rank** the keywords database. That means ordering all search phrases according with their frequency in the cache (database).
Your program should be able to print to a file the Top 1000 search phrases with their respective ranks (occurrence frequency).

E.g. Top 10 search phrases from **`keywords.db`** are:

```
real estate # 43298
for sale # 38022
new york # 27302
how to # 25068
web site # 21073
las vegas # 19039
cell phone # 17657
of the # 15012
credit card # 14278
web hosting # 11037
```

A ***search phrase*** as a pair of just two consecutive keywords in the query database.

E.g.
```
"cruise line"
"dell e153fp"
"cruise ship"
"samsung lcd"
"norwegian cruise"
```

STL for Competitive Programming and Software Development
Coding Test - April 2016

➽ Your second task is to implement your own auto-suggestion engine for 10 related searches, based on top search phrases containing the input keyword.
See previous *example* with suggested searches for keyword: "***cruise***".

This operation (user inputs a new keyword and the engine auto-suggests related search phrases) should be repetitive during a program session and should be superfast.

*** This interactive mode should be active only when the program receives a **`/search`** command-line switch.

➽ Provide an analysis of the runtime (steps) and space (memory usage) complexity in **Big-O** notation for all functions/codeblocks.

The analysis should include the ***average*** and ***worst-case* complexity** along with a brief explanation of your reasoning.
Write this information in comments; start them with: // [SPACE COMPLEXITY] or // [RUNTIME COMPLEXITY]

STL for Competitive Programming and Software Development
Coding Test - April 2016

Presenting his solution (the best),

Andrei Ciurez - student an I, Calculatoare Engleza

```
// Code walk-through
```

Course Evaluation:
"STL Algorithms - Principles and Practice" by CAPHYON

Please take the survey:

https://www.surveymonkey.com/r/JNBSZP7

CAPHYON