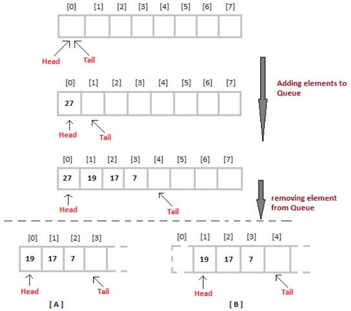
# STL - Principles and Practice

**Victor Ciura** - Technical Lead, Advanced Installer
**Gabriel Diaconița** - Senior Software Developer, Advanced Installer
http://www.advancedinstaller.com
**CAPHYON**

# Agenda

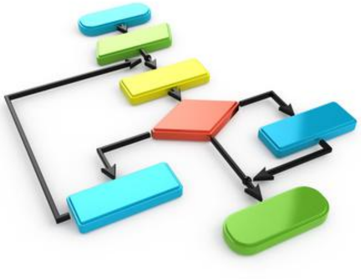
## Part 0: STL Intro.

## Part 1: Containers and Iterators



## Part 2: STL Function Objects and Utilities

## Part 3-4: STL Algorithms Principles and Practice



Part 4:

# STL Algorithms - Principles and Practice

*“Show me the code”*

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

```cpp
// 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 loop, 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;
```

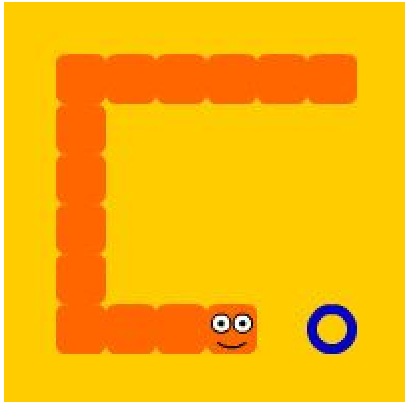
Implement a new API for the network, that efficiently finds a server node (address) containing a given data `Value`.

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

# Demo: Server Nodes

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

## Time for coding fun!



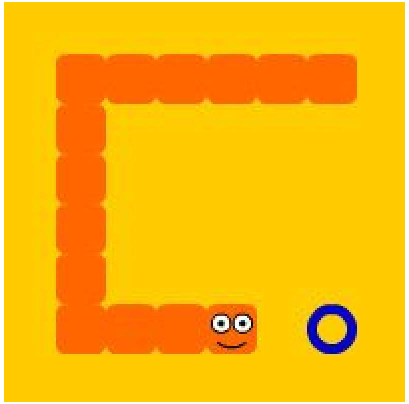
Our little game “**Worm STL”** it’s missing some key functionality.

Can you implement the required functionality using only STL algorithms?

Tell us your solution!

# Demo: Worm STL

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



# Course Evaluation:
# *"C++ STL - Principles and Practice"* by CAPHYON

**Please take the survey:**

[http://tinyurl.com/open4tech-stl](http://tinyurl.com/open4tech-stl)

[https://www.surveymonkey.com/r/open4tech-stl](https://www.surveymonkey.com/r/open4tech-stl)

SurveyMonkey®