Lessons Learned

Tony Van Eerd

May 12, 2015

How I Code and Why

Tony Van Eerd, Research In Motion

May 17, 2012

Examples That Stick/Stuck

Tony Van Eerd, Research In Motion

May 17, 2012

Lessons Learned

Tony Van Eerd

May 12, 2015

“Thanks”

(Blame)

inscriber TECHNOLOGY
Adobe
RIM
BlackBerry

```
{
    //
    // reads a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times.
    //

    if (argc < 3 || argc > 5) {
        return -1;
    }
    char const * intga = argv[1];
    char const * outtga = argv[2];

    int replicateX = argc >= 4 ? atoi(argv[3]) : 4;
    int replicateY = argc >= 5 ? atoi(argv[4]) : replicateX;

    TGAFileReader in(intga);

    static const int pixelSize = 4; // bytes per pixel - ie 32bpp
    //static const int replicate = 4;  // 4 x 4

    int dstWidth = in.getWidth() * replicateX;
    int dstHeight = in.getHeight() * replicateY;  // final height, not height of the dst buffer!

    // MUST do Bassamatic BEFORE Splunker
    bassamatic_init();
    splunker_init();

    char * dst = new char[dstWidth * in.getHeight() * pixelSize];  // buffer only needs to be sourceHeight high, and we will reuse 4 times
    char * dstStart = dst;
    int sourceLineByteLength = in.getWidth() * pixelSize;

    // read in image, replicating it across into 4 copies
    for (int y = 0; y < in.getHeight(); y++)
    {
        in.readLine(dst);
        // copy that line across 3 times, so we have it 4 times as wide
        for (int r = 1; r <= replicateX; r++)
        {
            std::memcpy(dst + r * sourceLineByteLength, dst, sourceLineByteLength);
        }
        dst += replicateX * sourceLineByteLength;
    }

    // now it is copied 4 times across, but still only 1x high

    if (in.isUpsideDown())
    {
        TGAFileFormat::flip_vert(dstStart, dstWidth, in.getHeight());
    }

    // now write out the 4x wide 4 times
    TGAFileWriter out(outtga, dstWidth, dstHeight);

    for (int z = 0; z < replicateY; z++)
    {
        out.writeLines(in.getHeight(), dstStart);
    }
```

```
{
    //
    // reads a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times.
    //

    if (argc < 3 || argc > 5) {
        return -1;
    }
    char const * intga = argv[1];
    char const * outtga = argv[2];

    int replicateX = argc >= 4 ? atoi(argv[3]) : 4;
    int replicateY = argc >= 5 ? atoi(argv[4]) : replicateX;

    TGAFileReader in(intga);

    static const int pixelSize = 4; // bytes per pixel - ie 32bpp
    //static const int replicate = 4;  // 4 x 4

    int dstWidth = in.getWidth() * replicateX;
    int dstHeight = in.getHeight() * replicateY;  // final height, not height of the dst buffer!

    // MUST do Bassamatic BEFORE Splunker
    bassamatic_init();
    splunker_init();

    char * dst = new char[dstWidth * in.getHeight() * pixelSize];  // buffer only needs to be sourceHeight high, and we will reuse 4 times
    char * dstStart = dst;
    int sourceLineByteLength = in.getWidth() * pixelSize;

    // read in image, replicating it across into 4 copies
    for (int y = 0; y < in.getHeight(); y++)
    {
        in.readLine(dst);
        // copy that line across 3 times, so we have it 4 times as wide
        for (int r = 1; r <= replicateX; r++)
        {
            std::memcpy(dst + r * sourceLineByteLength, dst, sourceLineByteLength);
        }
        dst += replicateX * sourceLineByteLength;
    }

    // now it is copied 4 times across, but still only 1x high

    if (in.isUpsideDown())
    {
        TGAFileFormat::flip_vert(dstStart, dstWidth, in.getHeight());
    }

    // now write out the 4x wide 4 times
    TGAFileWriter out(outtga, dstWidth, dstHeight);

    for (int z = 0; z < replicateY; z++)
    {
        out.writeLines(in.getHeight(), dstStart);
    }
```

```
{
    //
    // reads a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times.
    //

    if (argc < 3 || argc > 5) {
        return -1;
    }
    char const * intga = argv[1];
    char const * outtga = argv[2];

    int replicateX = argc >= 4 ? atoi(argv[3]) : 4;
    int replicateY = argc >= 5 ? atoi(argv[4]) : replicateX;

    TGAFileReader in(intga);

    static const int pixelSize = 4; // bytes per pixel - ie 32bpp
    //static const int replicate = 4;  // 4 x 4

    int dstWidth = in.getWidth() * replicateX;
    int dstHeight = in.getHeight() * replicateY;  // final height, not height of the dst buffer!

    // MUST do Bassamatic BEFORE Splunker
    bassamatic_init();
    splunker_init();

    char * dst = new char[dstWidth * in.getHeight() * pixelSize];  // buffer only needs to be sourceHeight high, and we will reuse 4 times
    char * dstStart = dst;
    int sourceLineByteLength = in.getWidth() * pixelSize;

    // read in image, replicating it across into 4 copies
    for (int y = 0; y < in.getHeight(); y++)
    {
        in.readLine(dst);
        // copy that line across 3 times, so we have it 4 times as wide
        for (int r = 1; r <= replicateX; r++)
        {
            std::memcpy(dst + r * sourceLineByteLength, dst, sourceLineByteLength);
        }
        dst += replicateX * sourceLineByteLength;
    }

    // now it is copied 4 times across, but still only 1x high

    if (in.isUpsideDown())
    {
        TGAFileFormat::flip_vert(dstStart, dstWidth, in.getHeight());
    }

    // now write out the 4x wide 4 times
    TGAFileWriter out(outtga, dstWidth, dstHeight);

    for (int z = 0; z < replicateY; z++)
```

```
{
    //
    // reads a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times.
    //

    if (argc < 3 || argc > 5) {
        return -1;
    }
    char const * intga = argv[1];
    char const * outtga = argv[2];

    int replicateX = argc >= 4 ? atoi(argv[3]) : 4;
    int replicateY = argc >= 5 ? atoi(argv[4]) : replicateX;

    TGAFileReader in(intga);

    static const int pixelSize = 4; // bytes per pixel - ie 32bpp
    //static const int replicate = 4;  // 4 x 4

    int dstWidth = in.getWidth() * replicateX;
    int dstHeight = in.getHeight() * replicateY;  // final height, not height of the dst buffer!

    // MUST do Bassamatic BEFORE Splunker
    // *otherwise* the splunker table…
    bassamatic_init();
    splunker_init();

    char * dst = new char[dstWidth * in.getHeight() * pixelSize];  // buffer only needs to be sourceHeight high, and we will reuse 4 times
    char * dstStart = dst;
    int sourceLineByteLength = in.getWidth() * pixelSize;

    // read in image, replicating it across into 4 copies
    for (int y = 0; y < in.getHeight(); y++)
    {
        in.readLine(dst);
        // copy that line across 3 times, so we have it 4 times as wide
        for (int r = 1; r <= replicateX; r++)
        {
            std::memcpy(dst + r * sourceLineByteLength, dst, sourceLineByteLength);
        }
        dst += replicateX * sourceLineByteLength;
    }

    // now it is copied 4 times across, but still only 1x high

    if (in.isUpsideDown())
    {
        TGAFileFormat::flip_vert(dstStart, dstWidth, in.getHeight());
    }

    // now write out the 4x wide 4 times
```

- *Thus…*

My favourite comment word is *Otherwise*.

- *Thus…*

My favourite comment word is *Otherwise*.

Hi Tony

How have you been?

I just wanted to thank you for impressing upon me the power of "otherwise <bad thing that happens>" in comments.

I employ otherwise often, and for that added information people have told me that they find my comments especially informative.

Sincerely,
…

- *Thus…*

My favourite comment word is *Otherwise*.

(“Why”)

```
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

```
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

Code Review *"the code appears to just getting a single value for … names. Where is loop (while or for) to iterate & get all the different … names and then update accordingly"*

```
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

Code Review *"the code appears to just getting a single value for … names. Where is loop (while or for) to iterate & get all the different … names and then update accordingly"*

Reply *"The concept behind this code and Jon's sample app, was that it identifies the problems one by one then updates one at a time. During the update the user will have to switch to … in the foreground. To solve certain issues, like the user selecting "OK" to the prompt, then existing out of …, the apps will have to implement this check in … and not in … in their … lifecycle. This means that everytime the app goes foreground after being backgrounded, the check is re-ran at which point the next issues will be identified and the next high priority update will prompt the user.."*

```
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

Code Review *"the code appears to just getting a single value for … names. Where is loop (while or for) to iterate & get all the different … names and then update accordingly"*

Reply *"The concept behind this code and Jon's sample app, was that it identifies the problems one by one then updates one at a time. During the update the user will have to switch to … in the foreground. To solve certain issues, like the user selecting "OK" to the prompt, then existing out of …, the apps will have to implement this check in … and not in … in their … lifecycle. This means that everytime the app goes foreground after being backgrounded, the check is re-ran at which point the next issues will be identified and the next high priority update will prompt the user.."*

?

```
// The concept behind this code and Jon's sample app, was that it identifies
// the problems one by one then updates one at a time. During the update the
// user will have to switch to … in the foreground. To solve certain issues,
// like the user selecting "OK" to the prompt, then existing out of …, the apps
// will have to implement this check in … and not in … in their … lifecycle.
// This means that everytime the app goes foreground after being backgrounded,
// the check is re-ran at which point the next issues will be identified and
// the next high priority update will prompt the user..“
//
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

Code Review *"the code appears to just getting a single value for … names. Where is loop (while or for) to iterate & get all the different … names and then update accordingly"*

Reply to Code Reviews *with Code*.

Also…

Comments tell *Why* not *What*.

```
// only process the first dependency;
// once it is taken care of, the background/foreground
// switch will automatically bring us back here,
// and the “next” one will be the new first one
// (as the old first one will have been updated and removed from the list)
//
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

```
// only process the first dependency;
// once it is taken care of, the background/foreground
// switch will automatically bring us back here,
// and the “next” one will be the new first one
// (as the old first one will have been updated and removed from the list)
//
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

```
// if we have any dependencies,
// update the first one
if (!dependencies.empty()) {
     auto name = dependencies.first();
     ...
     updateDependency(name);
}
```

Also…

Comments tell *Why* not *What*.

Reply to Code Reviews *with Code*.

Q
Q

```
class Card
{
    enum Suit { Hearts, Clubs, Diamonds, Spades };

    int rank;
    Suit suit;
    ...
};
```

Object

Value

Objects vs Values

- Object, QObject,…
- java / OOP
- non-copyable
- objects, things – *changeable*
- signals/slots – *observable*
- Relationships

- Steering Wheel Problem
- OH NO!!! Pointers!!!
- (Smart Pointers)
- (Qt – parent/child management)

- int
- Rect
- string
- copy
- Alex Stepanov
- Sean Parent
- John Lakos
- Math

- Oh, no pointers.

Objects vs Values

- Object, QObject,…
- java / OOP
- non-copyable
- objects, things – *changeable*
- signals/slots – *observable*
- Relationships

- Steering Wheel Problem
- OH NO!!! Pointers!!!
- (Smart Pointers)
- (Qt – parent/child management)

- int
- Rect
- string
- copy

Objects vs Values

- Object, QObject,…
- java / OOP
- non-copyable
- objects, things – *changeable*
- signals/slot
- Relationshi
- Steering W
- OH NO!!! P
- (Smart Poi
- (Qt – parent/child management)

```
y = x
print g(x)
print g(y)


print k(h(f(x), g(w)), h(f(x), g(w)))

z = h(f(x), g(w))
print k(z, z)
```

- int
- Rect
- string
- copy
- Alex Stepanov
- Sean Parent
- John Lakos
- Math
- Oh, no pointers.

Objects vs Values

Objects

- Object, QObject,…
- java / OOP
- non-copyable
- objects, things – *changeable*
- signals/slots – *observable*
- Relationships

- Steering Wheel Problem
- OH NO!!! Pointers!!!
- (Smart Pointers)
- (Qt – parent/child management)

Values

- int
- Rect
- string
- copy
- Alex Stepanov
- Sean Parent
- John Lakos
- Math

- Oh, no pointers.

Object

Value

```
class Card
{
    int rank;
    Suit suit;
    ...
};
```

Object

Value

No Liquids near Laptops?

Be Careful when Lending?

MAKE BACKUPS.

Object

Value

Object

```
class ...
{
    ...
};
```

Value

How to *NOT* call assert.

```
CountryAndCode countryCodes[] = {
   { “Afghanistan”, “AF”, “AFG” },
   { “Albania”, “AL”, ALB” },
   { “Algeria”, “DZ”, DZA” },
   { “Andorra”, “AD”, “AND” },
   …
};

static std::map<string, CountryAndCode> countryLookup;
static std::map<string, CountryAndCode> twoLetterLookup;
static std::map<string, CountryAndCode> threeLetterLookup;

void some_init()
{
    for (auto c : countryCodes) {
        countryLookup[c.country] = c;
        twoLetterLookup[c.twoLetter] = c;
        threeLetterLookup[c.threeLetter] = c;
    }
}
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        assert(is_sorted(begin, end));
    }

    template <typename Value> Iterator find(Value value)
    {
        // use binary search, as we know it is sorted
        …
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        assert(is_sorted(begin, end));
    }

    template <typename Value> Iterator find(Value value)
    {
        // use binary search, as we know it is sorted
        …
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        assert(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        MY_ASSERT(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        YOUR_ASSERT(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        YOUR_ASSERT(is_sorted(begin, end));
    }
};
```

Why do I want to call **your** assert? *(And who is 'I' and who is 'you'?)*

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        SORTED_VIEW_ASSERT(is_sorted(begin, end));
    }
};
```

```
#ifndef SORTED_VIEW_ASSERT
#define SORTED_VIEW_ASSERT  MY_ASSERT
#endif


template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        SORTED_VIEW_ASSERT(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        SORTED_VIEW_ASSERT(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        FRAMEWORK_ASSERT(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
        YOUR_ASSERT(is_sorted(begin, end));
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);
```

```
CountryAndCode countryCodes[] = {
   { “Afghanistan”, “AF”, “AFG” },
   { “Albania”, “AL”, ALB” },
   { “Algeria”, “DZ”, DZA” },
   { “Andorra”, “AD”, “AND” },
   …
};

sorted_view view = assert_sorted(countryCodes);

sorted_view view = assume_sorted(countryCodes);

sorted_view view = ensure_sorted(countryCodes);

sorted_view view = sort(countryCodes);
```

```
CountryAndCode countryCodes[] = {
#ifdef ABKHAZIA || …
   { “Abkhazia”, … },
#endif
   { “Afghanistan”, “AF”, “AFG” },
   { “Albania”, “AL”, ALB” },
   { “Algeria”, “DZ”, DZA” },
   { “Andorra”, “AD”, “AND” },
   …
};






sorted_view view = assert_sorted(countryCodes);

sorted_view view = assume_sorted(countryCodes);

sorted_view view = ensure_sorted(countryCodes);

sorted_view view = sort(countryCodes);
```

```
CountryAndCode countryCodes[] = {
#if …
   { “Abkhazia”, … },
#elif …
   { “Autonomous Republic of Abkhazia”, … },
#elif …
   { “Republic of Abkhazia”, … },
#endif
   { “Afghanistan”, “AF”, “AFG” },
   { “Albania”, “AL”, ALB” },
   { “Algeria”, “DZ”, DZA” },
   { “Andorra”, “AD”, “AND” },
   …
};


sorted_view view = assert_sorted(countryCodes);

sorted_view view = assume_sorted(countryCodes);

sorted_view view = ensure_sorted(countryCodes);

sorted_view view = sort(countryCodes);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(sort_certificate<Iterator,Sentinel,Less> cert)
        : begin_(cert.begin), end_(cert.end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sort_certificate
{
    Iterator begin_;
    Sentinel end_;

    sort_certificate(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sorted_view
{
    Iterator begin_;
    Sentinel end_;

    sorted_view(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sort_certificate
{
    Iterator begin_;
    Sentinel end_;

    sort_certificate(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);

sorted_view view = sort_certificate(some_begin, some_end);
```

```
template <typename Iterator, typename Sentinel, typename Less = blahblahblah>
struct sort_certificate
{
    Iterator begin_;
    Sentinel end_;

protected:
    sort_certificate(Iterator begin, Sentinel end) : begin_(begin), end_(end)
    {
    }
};
```

```
sorted_view view = assert_sorted(some_container);

sorted_view view = assume_sorted(some_container);

sorted_view view = ensure_sorted(some_container);

sorted_view view = sort(some_container);

//sorted_view view = sort_certificate(some_begin, some_end);
```

```
// pseudocode – TODO: templatize

sort_certificate ensure_sorted(Container container)
{

   bool sorted = is_sorted(container);
#ifdef NDEBUG
   if (!sorted) {
      YOUR_LOG("container not sorted");
      sort(container);
   }
#else
   YOUR_ASSERT(sorted);
#endif

   struct ensured_cert : sort_certificate
   {
      ensured_cert(Iterator begin, Sentinel end)
          : sort_certificate(begin,end)
      {
      }
   };

   return ensured_cert(container.begin, container.end);
}
```

```
// pseudocode – TODO: templatize

sort_certificate ensure_sorted(Container container)
{

   bool sorted = is_sorted(container);
#ifdef NDEBUG
   if (!sorted) {
      YOUR_LOG(“container not sorted”);
      sort(container);
   }
#else
   YOUR_ASSERT(sorted);
#endif

   struct ensured_cert : sort_certificate
   {
      ensured_cert(Iterator begin, Sentinel end)
          : sort_certificate(begin,end)
      {
      }
   };

   return ensured_cert(container.begin, container.end);
}
```

```
// pseudocode – TODO: templatize

sort_certificate ensure_sorted(Container container)
{
   // YOUR logic
   bool sorted = is_sorted(container);
#ifdef NDEBUG
   if (!sorted) {
      YOUR_LOG("container not sorted");
      sort(container);
   }
#else
   YOUR_ASSERT(sorted);
#endif

   struct ensured_cert : sort_certificate
   {
      ensured_cert(Iterator begin, Sentinel end)
          : sort_certificate(begin,end)
      {
      }
   };

   return ensured_cert(container.begin, container.end);
}
```

Adopter/Adapter.

Speaking of *ERRORS!!#$@*

```
if (ptr) {
   …
}
```

```
if (ptr) {
  …
}
else…
```

```
if (!ptr) {
  deal with it;
  return or throw;
}

// ‘normal’ code…
```

Error: An error occurs when a function can not complete its *primary* purpose.

Error: An error occurs when a function can not complete its *primary* purpose.

```
// returns (poor) approximation of square root
// returns -1 on error (eg negative input)
double square_root(double d)
{
    if (d < 0) {
       return -1;
    }
    return 17;
}
```

Error: An error occurs when a function can not complete its *primary* purpose.

```
// returns (poor) approximation of square root
// returns -1 on error (eg negative input)
double square_root(double d)
{
    if (d < 0) {
       return -1;
    }
    return 17;
}

// returns additive inverse of input
double negate(double d)
{
    return square_root(-22) * d;
}
```

Error: An error occurs when a function can not complete its *primary* purpose.

Bug: A bug is an error that can only be fixed by changing the code (or config file, etc…).

Error: An error occurs when a function can not complete its *primary* purpose.

Bug: A bug is an error that can only be fixed by changing the code (or config file, etc…).

Error vs Bug: Can't always tell which is which.

file-not-found on `open("/hardcoded/foo.bar");`

Expected vs Unexpected

ERRORS: Know your Audience.

Who do you need to notify about the error?

ERRORS: Know your Audience.

- function/library author (you)
- calling developer
- calling code
- end user

ERRORS: Know your Audience.

- function author (you)
- calling developer
- calling code
- end user

- All the above?

ERRORS: Know your Audience.

- function author (you)
  - assert/UB/crash/terminate/log
- calling developer
- calling code
  - throw/return/out-param
- end user

ERRORS: Know your Audience.

- function author (you) - unexpected
  my_assert/UB/crash/terminate/my_log
- calling developer - unexpected
  your_assert/UB/crash/terminate/your_log
- calling code - expected
  throw/return/out-param
- end user - expected
  message, etc

Testing:

M + N vs M x N

M + N vs M x N is for Unit Tests

Testing: Just do it.

That one simple rule to writing better code NOW!

Extra Slides…

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _disablePopupMenu = true;
    ...
    break;

case HOVER:
    if ( !_disablePopupMenu) {
        showPopupMenu();
    }
    break;

case UP:
    _disablePopupMenu = false;  // reset
    ...
    break;
```

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _movedSinceDown = true;
    ...
    break;

case HOVER:
    if ( !_movedSinceDown) {
        showPopupMenu();
    }
    break;

case UP:
    _movedSinceDown = false;  // reset
    ...
    break;
```

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _disablePopupMenu = true;
    ...
    break;

case HOVER:
    if ( !_disablePopupMenu) {
        showPopupMenu();
    }
    break;

case UP:
    _disablePopupMenu = false;  // reset
    ...
    break;
```

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _movedSinceDown = true;
    ...
    break;

case HOVER:
    if ( !_movedSinceDown) {
        showPopupMenu();
    }
    break;

case UP:
    _movedSinceDown = false;  // reset
    ...
    break;
```

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _disablePopupMenu = true;
    ...
    break;

case HOVER:
    if ( !_disablePopupMenu) {
        showPopupMenu();
    }
    break;

case UP:
    _disablePopupMenu = false;  // reset
    ...
    break;
```

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _disablePopupMenu = true;
    ...
    break;

case HOVER:
    if ( !_disablePopupMenu) {
        showPopupMenu();
    }
    break;

case UP:
    _disablePopupMenu = false;  // reset
    ...
    break;
```

Think about other code that needs to disable the popup menu.
Does it also set _disablePopupMenu?
or popupMenu.disable()?
who resets it?

```
case DOWN:
    ...
    break;

case MOVE:
    // disable popup menu for this touch sequence,
    // *otherwise* if we got a HOVER later (user stopped moving for a while)
    // then we would bring up the Menu,
    // and the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    _movedSinceDown = true;
    ...
    break;

case HOVER:
    if ( !_movedSinceDown) {
        showPopupMenu();
    }
    break;

case UP:
    _movedSinceDown = false;  // reset
    ...
    break;
```

Alternatively, think about other code that needs to set _movedSinceDown…

…Hopefully there is none!

```
case DOWN:
    ...
    break;

case MOVE:
    _movedSinceDown = true;
    ...
    break;

case HOVER:
    // the UX team says we don't want the popup menu to happen after a MOVE
    // (ie scroll then pause should not bring up the menu)
    if ( !_movedSinceDown) {
        showPopupMenu();
    }
    break;

case UP:
    _movedSinceDown = false;  // reset
    ...
    break;
```

```
case DOWN:
    ...
    break;

case MOVE:
    break;

case HOVER:
    break;

case DOWNHOVER:  // or some better name
    showPopupMenu();
    break;

case UP:
    ...
    break;
```

Thus…

“Separation of Concerns”

```
if ( !_disablePopupMenu)
```

Thus…

Avoid Double Negatives

```
class LockFreeList
{
public:
   bool isEmpty()   // or just empty()
   {
      ...
   }
};
```

```
{
   if (!list.isEmpty())
   {
      Foo foo = list.pop();
      ...
   }
};
```

```
class LockFreeList
{
public:
   bool wasEmpty()
   {
      ...
   }
};
```

Thus…

was not is
in threaded programming.