

How I Code and Why

Tony Van Eerd, Research In Motion

May 17, 2012



How do You 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



C++ Solution Station (?)

BoostCon/C++Now (2013?)





www.bobdevol.com

Single Responsibility Principle
Open/Closed Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle



“Thanks”



P.S. github.com/blackberry/Boost



```
// PWN a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times. May 17, 2012
//

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;

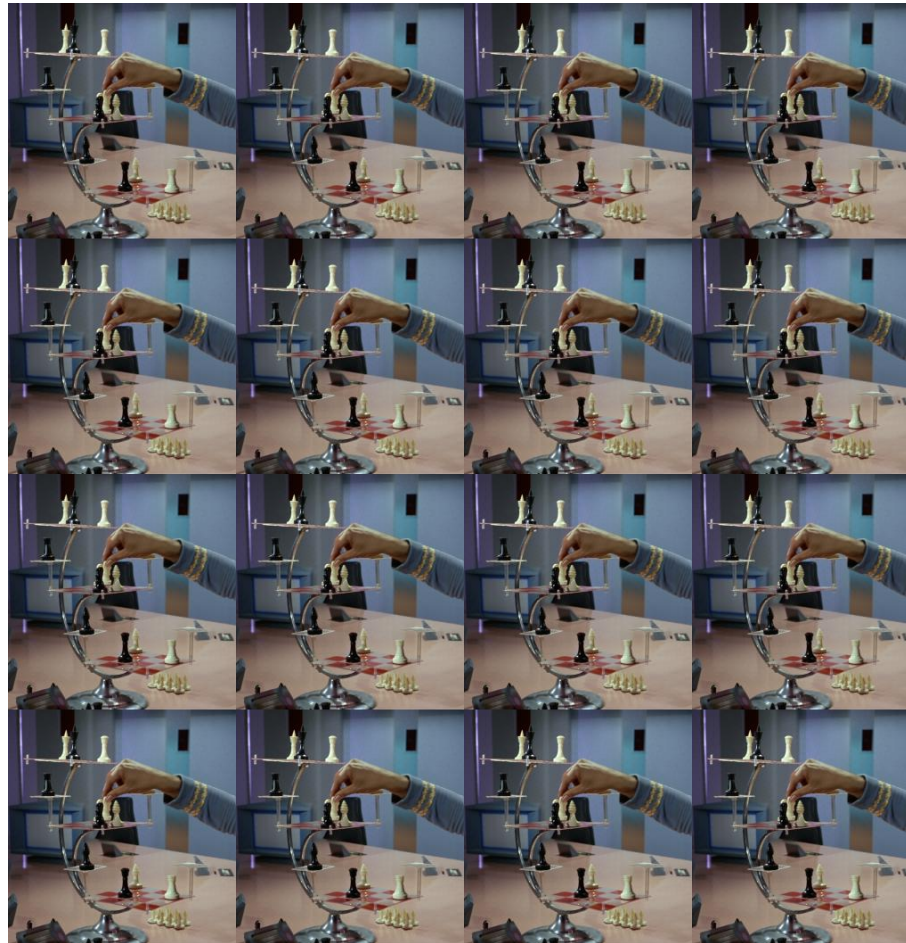
// read in image, replicating it across into 4 copies
for (int y = 0; y < in.getHeight(); y++)
{
    in.readLine(dst);
    // copy that line across 3 times, so we have it 4 times as wide
    for (int r = 1; r <= replicateX; r++)
    {
        std::memcpy(dst + r * sourceLineByteLength, dst, sourceLineByteLength);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



```
// PWN a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times.
// May 17, 2012

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);
}
```

```
t
// 
// Read a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times. May 17, 2012
//
```

```
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++)
```

```
t
//  a tga, writes out a tga with the image copied 4 times across and 4 times down (4x4) ie 16 times. May 17, 2012
//
```

```
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;

TGATFileReader 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())
{
    TGATFileFormat::flip_vert(dstStart, dstWidth, in.getHeight());
}

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

Thus...

My favourite comment word is
Otherwise.



```
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



How a Button invokes a 'click' action:

- `virtual Button::invokeAction()`
- `virtual Invokeable::invokeAction() // Button : private Invokeable {};`
- `(*invokeAction)(theirdata) // C styles`
- `_listener->invokeAction()`
- `boost::function`
- `"callable" // template<typename F> onClick(F f); // converts to function<> for you`
- `os/framework_sendmessage(destId, buttonId, actionId, etc)`
- `os/framework_postmessage(destId, buttonId, actionId, etc) /**`
- `queue a boost::function to a threaded work queue /**`
- `condvar /**`
- `boost::signal<>, Qt signal, framework signal`
- `member.invokeAction()` where `Button<T>` has a `T` member.
- `Base::invokeAction() // template <typename Base> Button : Base {};`
- `invokeAction()`



```
template <typename ActionFramework>
class Button : ActionFramework    // use CRTP?
{
    ...
    void handleInput(...)
    {
        if (...decide to invoke...)
        {
            invokeAction();
        }
    }
};
```

Thus...

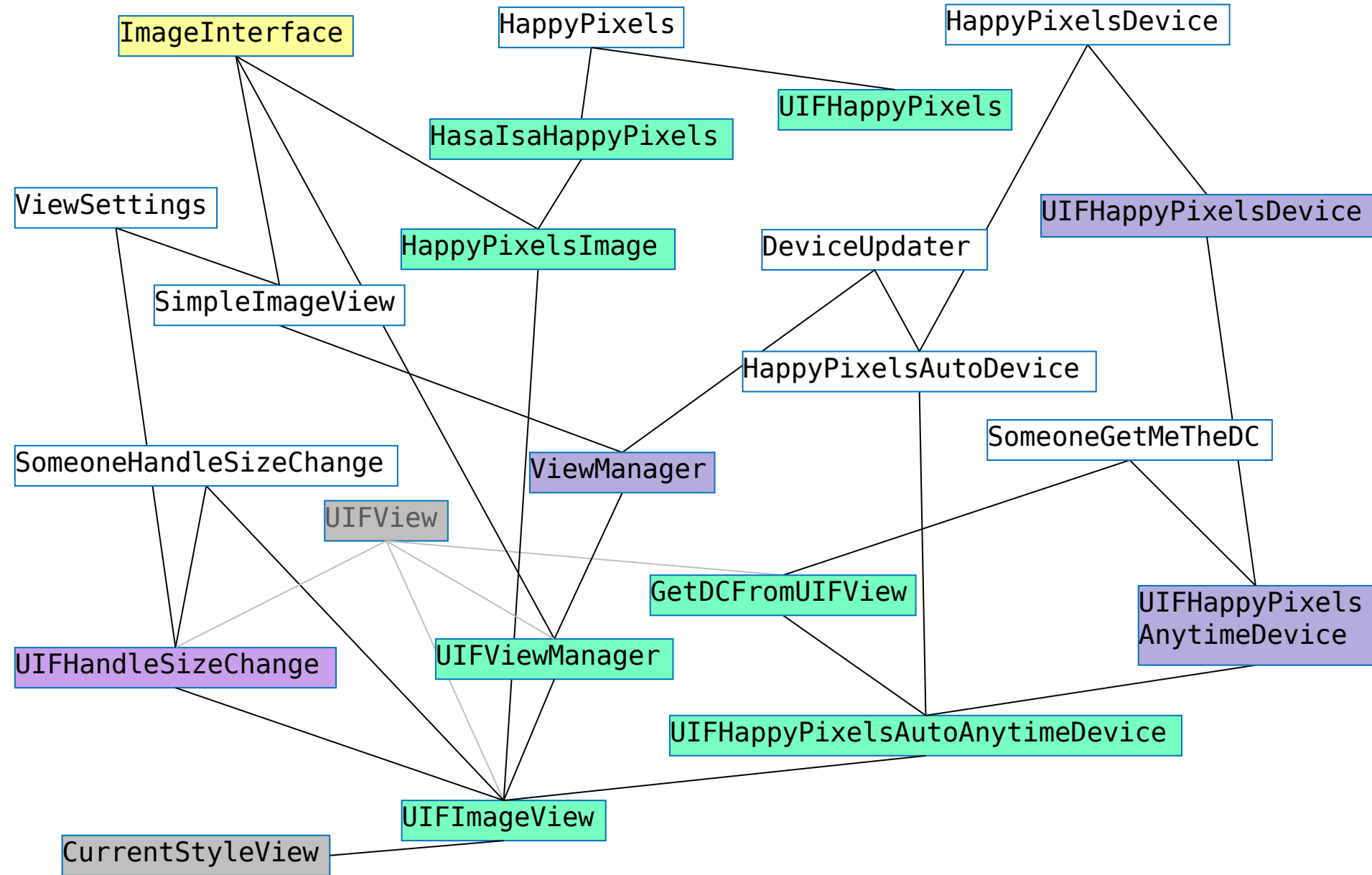
Separation of Concerns (?)

Inversion of Everything (?)

Top Down (?)

I don't care / not my problem





Thus...

?!



Speaking of Buttons...




```
class CheckBox
{
public:
    bool isChecked()
    {
        ...
    }
};
```

Sean Parent (paraphrased)

CheckBox::isChecked()
you're doing it wrong.



Speaking of Sean Parent...

No raw loops



How to go from Java to C++...

OH NO! Pointers!



How to go from Java to C++...

OH NO! Pointers!
Oh, No Pointers.



How to go from Java to C++...

OH NO! Pointers!
Oh, No Pointers.
Value Types.



Speaking of Sean Parent...

Value Types /
“shared_ptr is as good as a global”



Speaking of Sean Parent...

Value Types “shared_ptr



Speaking of 'is'...



```
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.



Also...

from not to.



$M + N$ vs $M \times N$



$M + N$ vs $M \times N$ is for Unit Tests



Examples That Suck

Tony Van Eerd, Research In Motion

May 17, 2012



“Structured Exception Handling”

(<http://msdn.microsoft.com/en-us/library/s58ftw19%28v=vs.80%29.aspx>)

MS Windows

```
--try
{
    // guarded code
}
--except ( expression )
{
    // exception handler code
}
```

“Structured Exception Handling”

(<http://msdn.microsoft.com/en-us/library/s58ftw19%28v=vs.80%29.aspx>)

MS Windows

```
__try
{
    // guarded code
}
__except ( expression )
{
    // exception handler code
}
```

Portable

```
OS_TRY
{
    // guarded code
}
OS_CATCH()
{
    // exception handler code
}
```

```
class OSCatcher
{
    static atomic<jmp_buf*> _current;

    jmp_buf _local, *_prev;
    bool _ok;

    OSCatcher() : _ok(true)
    {
        _prev = _current.exchange(&_local);
        if (setjmp(_local)) { // example! not thread safe
            _ok = false;
            _current = _prev;
        }
    }
    operator bool() { return _ok; }
};
```

```
class OSCatcher
{
    static atomic<jmp_buf*> _current;

    jmp_buf _local, *_prev;
    bool _ok;

    OSCatcher() : _ok(true)
    {
        _prev = _current.exchange(&_local);
        if (setjmp(_local)) { // example! not thread safe
            _ok = false;
            _current = _prev;
        }
    }
    operator bool() { return _ok; }
};

#define OS_TRY      if (OSCatcher catcher)
#define OS_CATCH() else
```

```
class OSCatcher
{
    static atomic<jmp_buf*> _current;

    jmp_buf _local, *_prev;
    bool _ok;

    OSCatcher() : _ok(true)
    {
        _prev = _current.exchange(&_amp;_local);
        if (setjmp(_local)) {
            _ok = false;
            _current = _prev;
        }
    }
    operator bool() { return _ok; }
};
```

```
#define OS_TRY      if (OS_Catcher catcher)
#define OS_CATCH() else
```

```
int main()
{
    signal(SIGINT, sigint_handler);
    ...
}

sigint_handler()
{
    longjmp(*OS_Catcher::_current, 1);
}
```

Thus...

Can != Should.




```
int func()
{
    static Once once;

    if (Once::Guard guard(once))
    {
        // init...
    }

    ...
}
```

```
#define once    static Once UNIQUE(once); \
                if (Once::Guard guard(once))

int func()
{
    once
    {
        // init...
    }

    ...
}
```

(Mostly unrelated actually...)

MACROS are evil



As Always...

Use Locks



Thus...

Experiment

Thank you for participating.

