A rant against flow control macros

devblogs.microsoft.com/oldnewthing/20050106-00

January 6, 2005

Raymond Chen

I try not to rant, but it happens sometimes. This time, I'm ranting on purpose: to complain about macro-izing flow control.

No two people use the same macros, and when you see code that uses them you have to go dig through header files to figure out what they do.

This is particularly gruesome when you're trying to debug a problem with some code that somebody else wrote. For example, say you see a critical section entered and you want to make sure that all code paths out of the function release the critical section. It would normally be as simple as searching for "return" and "goto" inside the function body, but if the author of the program hid those operations behind macros, you would miss them.

```
HRESULT SomeFunction(Block *p)
{
 HRESULT hr;
 EnterCriticalSection(&g_cs);
 VALIDATE_BLOCK(p);
 MUST_SUCCEED(p->DoSomething());
 if (andSomethingElse) {
  LeaveCriticalSection(&g_cs);
  TRAP_FAILURE(p->DoSomethingElse());
  EnterCriticalSection(&g_cs);
 }
 hr = p->DoSomethingAgain();
Cleanup:
 LeaveCriticalSection(&g_cs);
 return hr;
}
```

[Update: Fixed missing parenthesis in code that was never meant to be compiled anyway. Some people are so picky. – 10:30am]

Is the critical section leaked? What happens if the BLOCK fails to validate? If DoSomethingElse fails, does DoSomethingAgain get called? What's with that unused "Cleanup" label? Is there a code path that leaves the "hr" variable uninitialized?

You won't know until you go dig up the header file that defined the VALIDATE_BLOCK, TRAP_FAILURE, and MUST_SUCCEED macros.

(Yes, the critical section question could be avoided by using a lock object with destructor, but that's not my point. Note also that this function temporarily exits the critical section. Most lock objects don't support that sort of thing, though it isn't usually that hard to add, at the cost of a member variable.)

When you create a flow-control macro, you're modifying the language. When I fire up an editor on a file whose name ends in ".cpp" I expect that what I see will be C++ and not some strange dialect that strongly resembles C++ except in the places where it doesn't. (For this reason, I'm pleased that C# doesn't support macros.)

People who still prefer flow-control macros should be sentenced to maintaining the original Bourne shell. Here's a fragment:

```
ADDRESS alloc(nbytes)
    POS     nbytes;
{
    REG POS     rbytes = round(nbytes+BYTESPERWORD,BYTESPERWORD);
    LOOP    INT     c=0;
        REG BLKPTR  p = blokp;
        REG BLKPTR  q;
        REP IF !busy(p)
            THEN    WHILE !busy(q = p->word) DO p->word = q->word OD
                IF ADR(q)-ADR(p) >= rbytes
                THEN    blokp = BLK(ADR(p)+rbytes);
                    IF q > blokp
                    THEN    blokp->word = p->word;
                    FI
                    p->word=BLK(Rcheat(blokp)|BUSY);
                    return(ADR(p+1));
                FI
            FI
            q = p; p = BLK(Rcheat(p->word)&~BUSY);
        PER p>q ORF (c++)==0 DONE
        addblok(rbytes);
    POOL
}
```

Back in its day, this code was held up as an example of "death by macros", code that relied so heavily on macros that nobody could understand it. What's scary is that by today's standards, it's quite tame.

(This rant is a variation on one of my earlier rants, if you think about it. Exceptions are a form of nonlocal control flow.)

Raymond Chen

Follow