Crafting Self-Evident Code with D

-or-

How I figured out how to understand my own code

by Walter Bright
Dlang.org
August 2023
https://twitter.com/WalterBright

The highest praise for code:

“that's so simple,
anyone could have written it”

We've All Heard About

- Secure code
- Safe code
- Clean code
- Modern code
- Structured code
- Optimized code
- Clever code
- User friendly code
- Effective code
- Maintainable code

But Is Your Code Comprehensible?

```
#include <stdio.h>
#define O1O printf
#define OlO putchar
#define O10 exit
#define Ol0 strlen
#define QLQ fopen
#define OlQ fgetc
#define O1Q abs
#define QO0 for
typedef char lOL;

lOL*QI[] = {"Use:\012\011dump file\012","Unable to open file '\x25s'\012",
 "\012","    ",""};

main(I,Il) lOL*Il[];
{   FILE *L;
    unsigned lO;
    int Q,OL[' '^'0'],llO = EOF,

    O=1,l=0,lll=O+O+O+l,OQ=056;
    lOL*llL="%2x ";
    (I != 1<<1&&(O1O(QI[0]),O10(1011-1010))),
    ((L = QLQ(Il[O],"r"))==0&&(O1O(QI[O],Il[O]),O10(O)));
    lO = I-(O<<l<<O);
    while (L-l,1)
    {   QO0(Q = 0L;((Q &~(0x10-O))== l);
        OL[Q++] = OlQ(L));
        if (OL[0]==llO) break;
        O1O("\0454x: ",lO);
        if (I == (1<<1))
        {   QO0(Q=Ol0(QI[O<<O<<1]);Q<Ol0(QI[0]);
            Q++)O1O((OL[Q]!=llO)?llL:QI[lll],OL[Q]);/*"
            O10(QI[1O])*/
            O1O(QI[lll]);{}
        }
        QO0 (Q=0L;Q<1<<1<<1<<1<<1;Q+=Q<0100)
        {   (OL[Q]!=llO)? /* 0010 10lOQ 000LQL */
            ((D(OL[Q])==0&&(*(OL+O1Q(Q-l))=OQ)),
            OlO(OL[Q])):
            OlO(1<<(1<<1<<1)<<1);
        }
        O1O(QI[01^10^9]);
        lO+=Q+0+l;}
    }
    D(l) { return l>=' '&&l<='\~';
}
```

How I wrote code in the 1980s


ENG 1
ENG 2
OFF
GS192 TAS192
---°/---
TRK 339
33
30
ARPT
WPT
STA
WXR
TFC

Just Shoot Me Now

```
#define BEGIN {
#define END    }
```

Don't Reinvent bool

enum { No, Yes } // in my office, pls

enum { Yes, No } // no hire

Horrors Blocked By D

- Regex expressions with operator overloading
- Iostreams (I never remember which way the << goes)
- Metaprogramming with macros
- Argument Dependent Lookup
- SFINAE
- Floor wax or tasty dessert topping
- Multiple inheritance

Code flows from Left to Right
and Top to Bottom

(just like a book)

We do that already, right?

Oops

g(f(e(d(c(b(a))),3))))

UFCS To The Rescue

a.b.c.d(3).e.f.g;

Simpler Example of Left to Right

```
int a();
int b(int);

int oldway() => b(a);
int better() => a.b;
```

And Top to Bottom

```
import std.stdio;
import std.array;
import std.algorithm;

void main() {
    stdin.byLine(KeepTerminator.yes).
    map!(a => a.idup).
    array.
    sort.
    copy(stdout.lockingTextWriter());
}
```

The More Control Paths, the Less Understandable

Shaw : you know a great deal about computers, don't you?

Mr Spock : I know all about them.

Reduce Conditionals

```
version (X)
    doX();
doY();
if (Z)
    DoZ();
```

```
doX();
doY();
doZ();
```

Negation In English

Dr McCoy : We're trying to help you, Oxmyx.

Bela Oxmyx : Nobody helps nobody but himself.

Mr Spock : Sir, you are employing a double negative.

Negation in Code

if (!noWay)

Is inevitably perceived as

if (noWay)

Rewrite as a Positive

if (way)

Negation and version

version (!Windows) {...}

Is not allowed. But one can write:

version (Windows) else { … }

But why make it difficult?

Positives are Self Evident

```
version (Windows) { … }
else version (OSX) { … }
else static assert(“unsupported”);
```

DMD Hall of Shame

- tf.isnothrow
- IsTypeNoreturn
- Noaccesscheck
- Ignoresymbolvisibility
- Include.notComputed
- “not nothrow”

Compound If Conditionals

The following

```
if (A && B && C && D)
```

```
if (A || B || C || D)
```

Is far more comprehensible than

```
if (A && (!B || C))
```

De Morgan's Theorem to the rescue!

(!A && !B) => !(A || B)

(!A || !B) => !(A && B)

Mr Spock : Dazzling display of logic

From Ubuntu unistd.h

```
#if defined __USE_BSD || (defined \
__USE_XOPEN && !defined __USE_UNIX98)
```

Prof Marvel : I can't bring it back, I don't know how it works!

Casts Hide Bugs

- Make code harder to read
- Difficult to determine if casts are correct
- Sledgehammer
- Grep code for `cast`
- https://github.com/dlang/dmd/pull/15488

char* xyzzy(char* p)

- Does p modify what it points to?
- Is p returned?
- Does xyzzy free p?
- Does xyzzy save p somewhere, like in a global?

const char* xyzzy(return scope const char* p)

- p doesn't modify what it points to
- p is returned
- p is not free'd
- xyzzy doesn't squirrel away a copy of p

Payoff: things that don't need to be documented

Memory Allocation

- Memory allocated during a function should be free'd during that function, independent of caller
- Or pass allocator in as a parameter
- Have a “sink” parameter that accepts output

Pass Abstract “sink” For Output

```
import dmd.errors;
void gendocfile(Module m) {
    ...
    if (!success)
        error("expansion limit");
}
```

```
import dmd.errorsink;
void gendocfile(Module m, ErrorSink eSink) {
    ...
    if (!success)
        eSink.error("expansion limit");
}
```

https://github.com/dlang/dmd/pull/15471

Pass Files as Buffers Rather than Files to Read

```
void gendocfile(Module m, const(char)*[] docfiles) {
    OutBuffer mbuf;
    foreach (file; ddocfiles) {
        auto buffer = readFile(file.toDString());
        mbuf.write(buffer.data);
    }
    ...
}
```

```
void gendocfile(Module m, const char[] ddoctext) {
    ...
}
```

https://github.com/dlang/dmd/pull/15525

Move Calls to Environment to Caller

```
void gendocfile(Module m) {
    char* p = getenv("DDOCFILE");
    if (p)
        global.params.ddoc.files.shift(p);
}
```

https://github.com/dlang/dmd/pull/15503

Write to Buffer, Caller Writes File

```
void gendocfile(Module m) {
    OutBuffer buf2;
    ...
    writeFile(m.loc, m.docfile.toString(), buf2[ ]);
}
```

```
void gendocfile(Module m, ref OutBuffer outbuf) {

    ... // write to outbuf
}
```

https://github.com/dlang/dmd/pull/15535

Use Pointers to Functions (or Templates)

```
import dmd.doc;
bool expand(...) {
    if (isIDStart(p))
        ...
}
```

```
alias fp_t = bool function(const(char)* p);
bool expand(..., fp_t isIDStart) {
    if (isIDStart(p))
        ...
}
```

https://github.com/dlang/dmd/pull/15470

Two Categories of Functions

- Alter state of the program
  - doAction()
- Ask a question
  - isSomething()
  - HasCharacteristic()
  - These can hopefully be made pure

Try not to do both in one function. Makes it difficult to understand/modify it.

Line Things Up

```
final switch (of)
{
    case elf:       lib = LibElf_factory();         break;
    case macho: lib = LibMach_factory();    break;
    case coff:      lib = LibMSCoff_factory(); break;
    case omf:       lib = LibOMF_factory();     break;
}
```

Prof. Marvel : I have reached a cataclysmic decision!

Use ref Instead Of *

https://github.com/dlang/dmd/pull/15487

Takeaways

- Use language features as intended
- Avoid negation
- Left to right, top to bottom
- Functions do everything through front door
- Don't conflate engine with environment
- Reduce cyclomatic complexity
- Keep trying – this is a process!


HONDA