VEGA Systems
https://www.vega-net.co.jp/

技術勉強会
人に優しいコードを書こう

2023年1月11日

株式会社ベガシステム

古いネタですが…
全半角が混ざっていたり、タブ8だったり、0と""とNULLの区別が無かったり、
ネストが深かったりすると、プログラマを殺せると言われた時期がありました。

トップ › 2011年 › 6月 › 23日

2011年6月23日

プログラマを殺す方法

プログラマを殺す方法

プログラミング　プログラマー　ガイドライン

mitukiii 142884 70 14 B! 380 176 302

tatあt @_____t_t_
レッドブルの1.5Lペットボトルがでたら大量のプログラマが死ぬ
2011-06-23 00:30:55

(つ'ワ'c) く★しっぽ @ryosms
プログラマに何も言わずにレッドブルを渡すと死ぬ
2011-06-23 20:10:33

@MAENOX
プログラマをレッドブルで殴り続けると死ぬ
2011-06-23 20:42:45

Hideaki_npc @hgot07
プログラマは死ぬ が、ドクペを与えると生き返る。
2011-06-23 21:55:51

サテツ @fepfeil
プログラマに半角全角の入り乱れた数字を見せ続けると死ぬ。１５３６７９０２３４７７６５１０９９７５
2011-06-23 13:15:14

え？裏地ってつけなきゃいけないんですか？ @CDperson
半角全角入り交じった数字の羅列なんてプログラマじゃなくても発狂するのでは…
2011-06-23 14:26:59

あられ@創作アカ分け @turedure_arare
プログラマじゃなくても、全角を半角に直す作業が鬱すぎて死ぬ
2011-06-23 14:28:48

ミサゴさん @misagosan
全角半角入り乱れてるだけで死ぬ程度の奴はプログラマとは呼べない
2011-06-23 19:11:40

u_s_k @u_s_k
プログラマにハード8タブインデントを見せると死ぬ。
2011-06-23 19:54:11

dpp@FFXIV6.0クリア @dplusplus
プログラマにタブと半角スペースと全角スペースの入り交じった文字列を見せ続けると死ぬ
2011-06-23 18:53:05

ogami @Oogami_Zin
プログラマに全角スペース見せたら死ぬ
2011-06-23 20:19:48

《ネルヤ》というニャオスの指輪 @goldarn_ring
Pythonプログラマにタブとスペースが入り交じったインデントのプログラムを渡すと死ぬ
2011-06-23 18:54:35

くら @mizunokura
たとえ全角数字も全角スペースも通用しないプログラマでも、半角2つ・半角4つ・タブの三種類のインデントが混ざったソースを投げつけられると死ぬ。
2011-06-23 20:53:08

LYCHEE @lychee
プログラマに半角カナを強要すると死ぬ
2011-06-23 20:26:05

@toka0351
プログラマを発狂させるにはソース文に全角スペースをこっそり入れろ。バグは出るし簡単に探せないしうわあああああああああああああああああ
2011-06-23 20:36:36

Count_Zero @_Count_Zero
プログラマなら半角全角混じった数字見て死ぬくらいなら整形プログラム書くよ
2011-06-23 13:45:24

ARK=Wynn a.k.a. Elen E. @arkwynn
全角半角程度で死ぬプログラマは温室育ち。
2011-06-23 23:07:06

れぐたん @regtan
プログラマは刺身タンポポな仕事すると死ぬ。死ない奴はプログラマの基本的である「怠惰であること」が欠けている。
2011-06-23 22:02:39

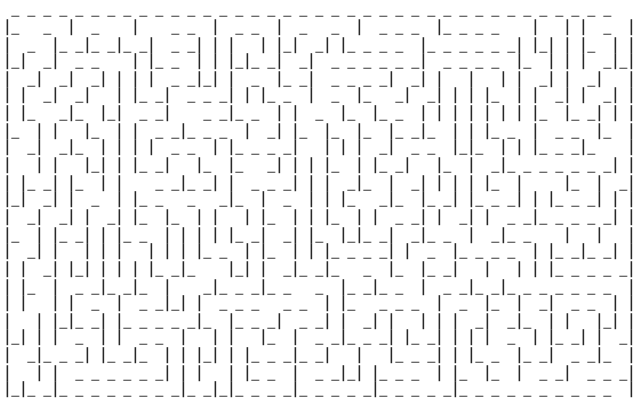
Ichiro S* @ichiro_s
プログラマに0と""とNULLの区別のない構文を見せると死ぬ
2011-06-23 20:09:26

わかめ@毎日猫がいる @vvakame
プログラマに7重ネストループ見せると死ぬ
2011-06-23 19:58:25

0．はじめに

このC言語プログラムを実行すると、同じ結果が得られます。
左は面白いけど、読みにくいですよね。
世の中には、プログラムの読みにくさを競うコンテストがあります。
「The International Obfuscated C Code Contest」

```
#define P(X)j=write(1,X,1)
#define C 39
int M[5000]={2},*u=M,N[5000],R=22,a[4],l[]={0,-1,C-1,-1},m[]={1,-C,-1,C},*b=N,
*d=N,c,e,f,g,i,j,k,s;main(){for(M[i=C*R-1]=24;f|d>=b;){c=M[g=i];i=e;for(s=f=0;
s<4;s++)if((k=m[s]+g)>=0&&k<C*R&&!(s[l]!=k%C&&(!M[k]||!j&&c>=16!=M[k]>=16))a[f++
]=s;if(f){f=M[e=m[s=a[rand()/(1+2147483647/f)]]+g];j=j<f?f:j;f+=c&-16*!j;M[g]=
c|1<<s;M[*d++=e]=f|1<<(s+2)%4;}else e=d>b++?b[-1]:e;}P(" ");for(s=C;--s;P("_"))
P(" ");for(;P("¥n"),R--;P("|"))for(e=C;e--;P("_ "+(*u++/8)%2))P("| "+(*u/4)%2
);}
```



```
#define P(X)j = write(1,X,1)
#define C 39

int M[5000] = {2};
int *u = M;
int N[5000];
int R = 22;
int a[4];
int l[] = {0, -1, C-1, -1};
int m[] = {1, -C,  -1,  C};
int *b = N;
int *d = N;
int c, e, f, g, i, j, k, s;

main()
{
    for(M[i=(C*R)-1] = 24; f|d >= b; )
    {
        c = M[g=i];
        i = e;
        for(s = f = 0; s < 4; s++)
        {
            if((k = m[s]+g) >= 0 &&
               k < C*R &&
               l[s] != k % C &&
               (!M[k] || !j && c >= 16 != M[k] >= 16))
            {
                a[f++] = s;
            }
        }

        if(f)
        {
            f = M[e = m[s = a[rand() / (1+2147483647/f)]] + g];
            j = j < f ? f : j;
            f += c &-16 * !j;
            M[g] = c | 1 << s;
            M[*d++ = e] = f | 1 << (s+2) % 4;
        } else {
            e = d > b++ ? b[-1] : e;
        }
    }

    P(" ");

    for(s=C; --s; P("_"))
    {
        P(" ");
    }

    for( ; P("¥n"), R--; P("|"))
    {
        for(e=C;e--;P("_ "+(*u++/8)%2))
        {
            P("| "+(*u/4) % 2);
        }
    }
}
```

人に優しいコード、
数日後、数週間後の自分が理解できるコード、
書けていますか？

今回のテーマは「人に優しいコードを書こう」です。
理解しやすい・可読性の良いコードの書き方 を考え直せればと思います。

- 良い名前をつける
- 適切なコメントを書く
- 意味のある単位に分割する
- キレイに整形する

プログラムに使われる名前は、主に次の5つの鉄則を守る必要がある。

1. 明確な単語を選ぶ
2. 汎用的な名前を避ける
3. 抽象的な名前よりも具体的な名前を使う
4. 接尾辞や接頭辞を使って情報を追加する
5. 名前の長さを決める

1. 明確な単語を選ぶ

```
class Thread {
    void Stop();
};
```

これでも良いが、動作に合わせてもっと明確な名前にした方が良い。
・取消ができない重い操作なら Kill()
・後から再開 Resume() できるなら、一時停止 Pause()

2. 汎用的な名前を避ける

```
def func1(value):
    return value ** 2
```

valueの2乗の値を示しているので、
関数名は func1 ではなく square などとした方が良い。

3. 抽象的な名前よりも具体的な名前を使う

任意のTCP/IPポートをサーバがリッスンできるかを確認する
ServerCanStart() という名前のメソッドがあったとする。
より具体的な名前としては CanListenOnPort() とした方が良い。

4. 接尾辞や接頭辞を使って情報を追加する

時間やバイト数のように計測できる値であれば変数名に
単位を入れた方が良い。

関数の仮引数	単位を追加した仮引数
Start(int delay)	delay → delay_secs
CreateCache(int size)	size → size_mb
ThrottleDownload(float limit)	limit → max_kbps
Rotate(float angle)	angle → degrees_cw

5. 名前の長さを決める

変数や関数の名前は「長い名前を避ける」という暗黙の了解がある。
これを真に受けると、1つの単語や1文字だけの名前になってしまう
場合があるが、スコープが小さければ短い名前でも構わない。

```
if (debug) {
    map<string, int> m;
    LookUpNamesNumbers(&m);
    Print(m);
}
```

データベースの問い合わせ結果を処理するコードを書いているとする。
results = Database.all_objects.filter(“year <= 2011”)

filterがあいまいな言葉であるため、
resultsがどちらを指しているのか分からない。
・「year <= 2011」のオブジェクト
・「year <= 2011」ではないオブジェクト

「選択する」のであれば select()
「除外する」のであれば exclude() にした方が良い。

```
#define P(X)j=write(1,X,1)
#define C 39
int M[5000]={2},*u=M,N[5000],R=22,a[4],l[]={0,-1,C-1,-1},m[]={1,-C,-1,C},*b=N,
*d=N,c,e,f,g,i,j,k,s;main(){for(M[i=C*R-1]=24;f|d>=b;){c=M[g=i];i=e;for(s=f=0;
s<4;s++)if((k=m[s]+g)>=0&&k<C*R&&l[s]!=k%C&&(!M[k]||!j&&c>=16!=M[k]>=16))a[f++
]=s;if(f){f=M[e=m[s=a[rand()/(1+2147483647/f)]]+g];j=j<f?f:j;f+=c&-16*!j;M[g]=
c|1<<s;M[*d++=e]=f|1<<(s+2)%4;}else e=d>b++?b[-1]:e;}P(" ");for(s=C;--s;P("_")
)P(" ");for(;P("¥n"),R--;P("|"))for(e=C;e--;P("_ "+(*u++/8)%2))P("| "+(*u/4)%2
);}
```

IOCCC日本語ネタバレ解説
https://mame.github.io/ioccc-ja-spoilers/

条件やループなどの制御フローはできるだけ「自然」にする。
コードの読み手が立ち止まったり読み返したりしないように書く。

どちらの方が読みやすいか？
・if (length <= 10)
・if (10 >= length)

最初の方が読みやすい理由

左側	右側
「調査対象」の式。変化する。	「比較対象」の式。あまり変化しない。

■質疑応答で出た例
if (NULL == a)
if (NULL = a)　コンパイルエラー

if (a == NULL)
if (a = NULL)

if (0 <= length && length <= 10)
0 <= length <= 10

三項演算子も使いどころに注意が必要。

○前者の方が読みやすい。後者は長くて冗長に感じる。

```
time_str += (hour >= 12) ? “pm” : “am”;
```

```
if (hour >= 12) {
    time_str += “pm”;
} else {
    time_str += “am”;
}
```

×無理やり1行に納められていて、読みづらい。

```
return exponent >= 0 ? Mantissa * (1 << exponent) : Mantissa / (1 << -exponent);
```

こうすると、多少マシになるけど。

```
return exponent >= 0 ?
    Mantissa * (1 << exponent) :
    Mantissa / (1 << -exponent);
```

変数のことが見える行数をできるだけ減らすと良い。

- 関数から早く返す

```
public Boolean Contains(String str, String substr) {
    if (str == null || substr == null) return false;
    if (substr.equals("")) return true;
    …
}
```

- ネストを浅くする
  右の方がネストが浅くて読みやすい。

```
if (user_result == SUCCESS) {
    if (permission_result != SUCCESS) {
        reply.WriteErrors(permission_result);
        reply.Done();
        return;
    }
    reply.WriteErrors("");
} else {
    reply.WriteErrors(user_result);
}
reply.Done();
```

```
if (user_result != SUCCESS) {
    reply.WriteErrors(user_result);
    reply.Done();
    return;
}

if (permission_result != SUCCESS) {
    reply.WriteErrors(permission_result);
    reply.Done();
    return;
}

reply.WriteErrors("");
reply.Done();
```

2.2.巨大な式を分割する

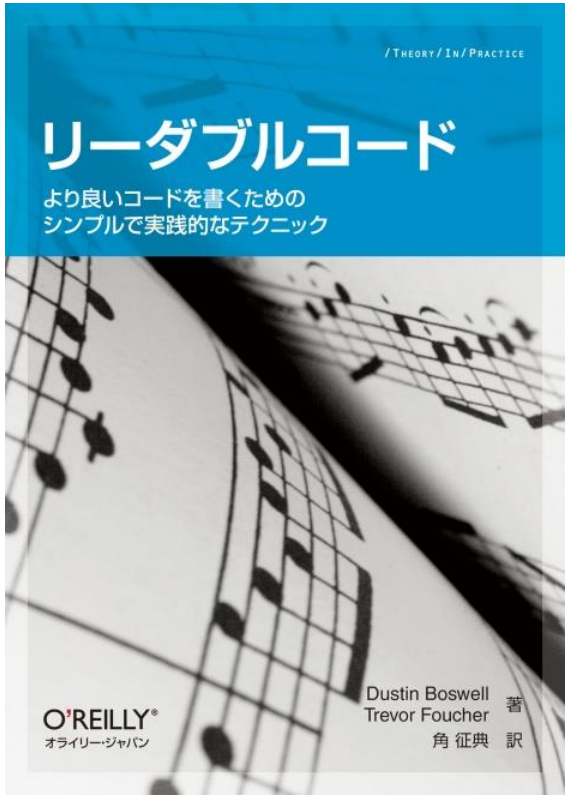
```
var update_highlight = function (message_num) {
    if ($("#vote_value" + message_num).html() === "Up") {
        $("#thumbs_up" + message_num).addClass("highlighted");
        $("#thumbs_down" + message_num).removeClass("highlighted");
    } else if ($("#vote_value" + message_num).html() === "Down") {
        $("#thumbs_up" + message_num).removeClass("highlighted");
        $("#thumbs_down" + message_num).addClass("highlighted");
    } else {
        $("#thumbs_up" + message_num).removeClass("highighted");
        $("#thumbs_down" + message_num).removeClass("highlighted");
    }
};
```

```
var update_highlight = function (message_num) {
    var thumbs_up   = $("#thumbs_up" + message_num);
    var thumbs_down = $("#thumbs_down" + message_num);
    var vote_value  = $("#vote_value" + message_num).html();
    var hi = "highlighted";

    if (vote_value === "Up") {
        thumbs_up.addClass(hi);
        thumbs_down.removeClass(hi);
    } else if (vote_value === "Down") {
        thumbs_up.removeClass(hi);
        thumbs_down.addClass(hi);
    } else {
        thumbs_up.removeClass(hi);
        thumbs_down.removeClass(hi);
    }
};
```

同じ式を要約変数として関数の最上部に抽出すると良い。

エンジニアは絶対読みましょう。



英語でも大丈夫！って人なら無料で読めます。
https://mcusoft.files.wordpress.com/2015/04/the-art-of-readable-code.pdf