バッドコード
福井技術者の集いその3 2015

自己紹介

- 緑川 志穂(@elliptic_shiho)
- 京都の奥地に生息する高校1年生
- プログラミング, 数学(楕円関数論, 複素解析, 暗号学, ...)
- 綺麗なコードの話題を前回LTしました

バッドコード

- 今回はC言語/JVM言語をベースに話します
- どんな言語/環境でも適用できる内容のはず

バッドコード

- 見た目が汚いコード
- コンパイル時の最適化が効かないコード
- "無駄な"細分化
- 依存
- 握りつぶし

バッドコードで何が悪い?

- 汚い(確信)
- 想定外の動作によるデバッグの工数増加
- 高速化したくても一定以上が難しくなる
- 大体保守性が悪い

見た目悪い

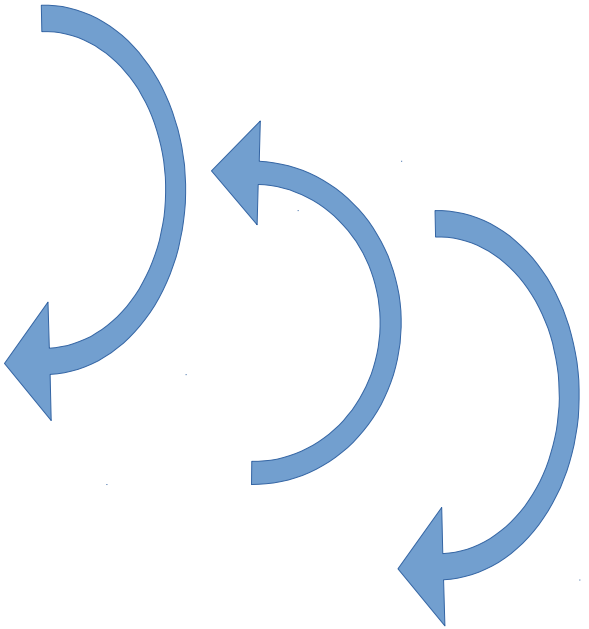
- スパゲッティコードは代表格
- 複雑すぎるコード
- 言語仕様/演算子の優先順位を使ったコード
  - ひと目で見て分からないのがボーダー
  - `for(char *p=s;d[p-s]=*p;p++);`とか

見た目悪い

```
int main (int ac, char **av) {
  puts("0");goto L2;
L1:
  puts("2");goto END;
L2:
  puts("1");goto L1;
END:
  puts("3");return 0;
}
```

見た目悪い

```
int main (int ac, char **av) {
  puts("0");goto L2;
L1:
  puts("2");goto END;
L2:
  puts("1");goto L1;
END:
  puts("3");return 0;
}
```



最適化が効かない

- defineを定数ポインタに置き換えるとか[次頁参照]
  - const char * constを使いましょう
- 変に自力で最適化もどきをする
  - pow(2, 10)を2<<9にするとか
  - 自分がコンパイラよりも優秀だと思うならどうぞ
  - 多くのコンパイラは綺麗なコードほどより高速化する設計

最適化が効かない

```
/* define */

#define HOGE "hoge" // 見たまんま定数

puts(HOGE);


/* const char * */

const char *HOGE = "hoge"; // ポインタの示す先が変わらない保証がこのコードだけではできない

puts(HOGE);


/* const char * const */

const char * const HOGE = "hoge"; // 示す先も示す先の内容も変わらないことが保証されている

puts(HOGE);
```

最適化が効かない

- リフレクションの多用
  - 実行時のメモリ読み書き量の増加
  - 使わないに越したことはない
- 変なバイトコード操作(~~黒魔術~~)
  - JITコンパイラが優秀になってきたから最近は大丈夫...?

依存

- gcc/VC++の独自拡張への依存
  - nested-function, strcpy_s, #pragma once...
  - OSSなら環境限定しない限りは避けたい
- OS特有の関数群、特定アーキ限定のコード等
  - 汎用部分と依存部分を分離すべき
  - Linux Kernelは好例

無駄な細分化

- クラス分け・関数化が細かすぎること
- JVM言語だと切実
  - クラスごとにファイルが生成される = ロードの分のオーバーヘッド
  - Androidアプリケーション作るなら気にした方がいいかもしれない

握りつぶし

- 開発やテストでは大体の現場ではツールを使う
  - JUnit, clangの静的解析機能, javac, ...
- 特に例外未キャッチの場合、"取りあえず"ということで以下のようなコードを書く人が...

```
try {
  hogehoge();
} catch(Exception e) {
  ; // 黙らせるためのセミコロン
}
```

握りつぶし

- 開発やテストでは大体の現場ではツールを使う
  - JUnit, clangの静的解析機能, javac, ...
- 特に例外未キャッチの場合、"取りあえず"ということで以下のようなコードを書く人が...

```
try {
  hogehoge();
} catch(Exception e) {
  ; // 黙らせるためのセミコロン
}
```

握りつぶし

- 何がダメ?
  - Developmentビルドではキャッチをしてくれるかもしれないが、Releaseビルドだと異常終了になる
- その結果↓

???「おたくの製品、なんか落ちるんだけど!」

  - せめてエラー画面をちゃんと作りましょう

握りつぶし

- 例外未処理はひとつの例
  - ツールの精度も上がってるので、結構欠陥は検出されやすくなっている
- 仕様でそのコードを修正できない or それでないとだめならコメントで記載する
  - 他人が見ても分かるように
    - 一人で開発しているわけじゃない

対策

- コーディング規約を文書等で明示化
- バッドコードをあえて書いてみる
  - プロダクトコード以外で
- 開発初期で制約がないのであれば言語を変える
- 余裕があれば設計段階からの見直し

コードゴルフ

- どれだけ短いコードを書くか
- メリット
  - 言語を学ぶ早道
  - 特殊文法を使ったりする事もあるから単純に楽しい(特にPerl)
- デメリット
  - 依存性も綺麗さもへったくれもない
  - サイズ削減等の目的がない限りプロダクトコードでやるのはダメ

Polyglot

- 2つ以上の言語で実行/コンパイルできるようなコードを書く
- メリット
  - 楽しい(小並感)
  - パズルをやる感覚
- デメリット
  - コードゴルフに同じ

何が言いたいのか

- コードゴルフ/Polyglotのコードは大概バッドコードに当てはまる
- そうでもしないと学ばないような仕様を学べる
  - 知識は多い方が良い

何が言いたいのか

- ワンライナーぐらい書けるようになっておくと作業効率は上がる(かも知れない)
- 短いコード=速いアルゴリズムなことも多い
  - 競プロで短くしてたらTLE->ACなんてことも

まとめ

- 綺麗なコードかつバッドコードは十分にありえる
- コーディングに関する知識だけでなく、環境に関する知識を持とう
- たとえ小規模なコードだったとしても、大規模化を十分に想定したコードを書こう

参考資料

- Software Design 2015/04 第一特集 トラブルシューティングの極意 技術評論社
- C言語 デバッグ完全解説 坂井丈泰/坂井弘亮著 技術評論社
- Code Golf http://shinh.skr.jp/dat_dir/golf_prosym.pdf
- 他多数

ありがとうございました