綺麗なコードを書くために

福井技術者の集いその2 2015

自己紹介

- 緑川 志穂(@elliptic_shiho)
- 普通の高校受験生
- プログラミングはまだ4〜5年
- 数学(複素解析/楕円関数論他), 暗号学
- OS/CPU作るところからWebアプリまで
- 最近はCTFも

綺麗なコード
#とは

綺麗なコードとは

- 可読性 のあるコード
- 移植性 のあるコード
- 保守性 のあるコード

可読性

- わかりやすい
- 比較的実現が簡単
- 一目で何をやっているのか分かるのが理想
- CTF/競プロでは気にしない

移植性

- クロスプラットフォームの意味だけでは無い
  - コードの流用
- 現場では流用は日常茶飯事
  - OSSでもある
- 必ずしも必要なわけではないが、考えておくべき

保守性

- 陰ながら最重要
- 経験積んで覚える部分が大きい
- これのないコードしか書けない⇒ずっと続くプロダクトはいつまでも書けない
  - 一発屋で生きていけるなら別に良い(?)

要するに

- 綺麗に書くことはソリューションの一定のクオリティの担保にもつながる
- 好感を持たれやすい
- バグ探しもやりやすい

→普段からも綺麗に書こう

綺麗に書くためには?

- コードを読む
  - 綺麗でも汚くてもいい
  - 汚ければ反面教師
  - もちろん綺麗なコードは読むべき
  - デザインパターンも良い
- 汚いもの知らないのに綺麗なものが書けるわけがない

綺麗に書くためには?

- その言語をちゃんと理解する
  - Cなら頭の中で大方アセンブル出来るぐらいまで
  - 応用まで出来ない言語なら無理に綺麗に書く必要はない
- どれだけ慣れてる言語でも手元にリファレンス一冊置いときましょう

綺麗に書くためには?

- オブジェクト指向/MVCはちゃんと理解する
  - いくら強い武器でも使い方が分からないとただのガラクタ
- 無駄な細分化はしない
  - 多すぎるとゴチャゴチャする&逆にオーバーヘッドに
- 様々な言語、様々な実装パターンを知る
  - 例えばC/C++/Scala/Lispで同じプログラムを書くとか
  - 多角的な見方を持つ

綺麗に書くためには?

- 客観的な意見を(貰う|持つ)
  - ペアプログラミングやコードレビューはひとつの手
  - 一人じゃ気づけない実装の不備/改善点が見つかる事も
  - 工期ヤバいなら無理はダメゼッタイ

コメント

- あればあるほどいいというものでもない
  - コメント7割にコード3割とかだと本末転倒
  - /* ○○処理 */ぐらいなら関数化したほうがいい
- 適度には必要
  - トリッキーな、ひと目で見てわからない部分等
  - 何か問題点がある時の一時的なTODOとして
    - 開発者間の連絡や日記には使わない

チーム開発

- 各々が技術があったとしても、協調性を持って統一したコードを書けないなら宝の持ち腐れ
  - 規約は†宗教戦争†になる前に文書化
  - 相手のコーディングスタイルを受け入れる事も大事

チーム開発

- 一人での開発以上に分かりやすいコーディングを心がける
- 整合性の取れたコードを書く
  - 個人個人のVCSの知識
  - プロジェクトリーダー/マネージャの技能

KISSの原則

- Keep It Simple, Stupid(シンプルにしておけ、愚か者)
- すごく大事
- シンプルに書くと自動的に見やすくもなる
- 必要外の処理をどれだけ減らせるかとも言える

YAGNI

- You aren't going to need it(あなたは多分それを必要としない)
- エクストリーム・プログラミングでの考え方
- 今必要としないのであれば、後で書けばいいという発想
- 後から流用されても良いコードを書くという意味もある

その他

- 綺麗なコードだからといってバグが無いわけではない
  - 人間が書く以上必ずあるから仕方がない
  - テスト項目にも気を抜かない
  - デバッグ用マクロを定義するのも手

デバッグ用マクロ例(C)

```
#ifdef NDEBUG
# define CHK_DEBUG()
# define DEBUG(x, ...)
#else
# define CHK_DEBUG() fprintf(stderr, "%s:%d <%s>\n", __FILE__,
__LINE__,__FUNCTION__)
# define DEBUG(x, ...) do {\
  fprintf(stderr, "%s:%d <%s> -> ", __FILE__, __LINE__, __FUNCTION__);\
  fprintf(stderr, x, __VA_ARGS__); \
  fprintf(stderr, "\n"); \
} while (0)
#endif /* end of NDEBUG */
```

その他

- デバッガブルなコードと綺麗な(特にコンパクトな)コードは大抵相容れない
  - 一行にまとめたコード等はステップ実行してるとなかなか本処理に入れなくてイライラする
  - トレードオフ

その他

- デバッガブルなコード

```
int i = a();
int j = b();
int k = c();
func(i, j, k);
```

- 綺麗な(まとまっている)コード

```
func(a(), b(), c());
```

まとめ

- いろいろな人の書いたコードを読む(ex, Linux, gcc, ffmpeg, ...)
- 毛嫌いはNG、オールマイティに情報を吸収していくのが大事
- PHPとアセンブラを同列に持ってくるぐらいの知識の幅とゆとりを持ったコーディングをしよう
- 経験を積む(超重要)
  - デバッグにも役立つ
  - 特に失敗する経験は重要
- 楽しもう

蛇足

- もし詰まったら一旦寝るか本でも読もう
  - 落ち着いて見直すと欠点に気づきやすい
  - 最近の方であればアニメでもいいかも
- リフレッシュの時間を確保する事

参考文献

- 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
- デバッグの理論と実践 - Andreas Zeller 著 中田 秀基 監訳 オライリー・ジャパン
- プログラマのための硬派なコンピュータ学 - 清野 克行 著(絶版) 秀和システム
- リバースエンジニアリングバイブル ~コード再創造の美学~ - 姜 秉卓 著 インプレスジャパン
- その他たくさん

ありがとうございました。