ソフトウェア設計及び実験 第2回
プログラミング作法

松本 和幸

2019年4月16日

1　プログラミング作法とは

プログラミング作法とは,「ソースコードを書く際の決め事」のことである．プログラミングスタイルとも呼ぶ．開発者の好みのスタイルで書いて良い場合や，会社の標準で決められた作法を守らなければならない場合もある．しかし，あくまで作法なので，決まり事を守らなくても文法さえ正しければ動作してしまう．このことが，後にデバッグしにくくなるコードを産み出す要因となる．プログラミング作法はコードの保守に欠かせない概念である．他人に読みやすいコードを書くことは，同時に自分にとっても読みやすいコードを書くことにもなるため，ぜひとも身につけてもらいたい．

2　コードの見た目

良いコードと悪いコードの違いについて述べる．たとえば，インデントが適切でなかったり，変数や定数と演算子との間にスペースを入れていなかったり，改行が無くブロックが分かりづらいコードなどは，一般的に見やすい（良い）コードではない．読みやすさというのは個人差があるため，明確な決め事は無いが，良いコードを書くために，以下の点について注意してほしい．

- インデント
  字下げのこと．適切におこなうことで，コードを見やすくすることができる．たとえば多重ループや入れ子構造など構文が複雑な場合にインデントによりブロック単位をわかりやすくすることなどが挙げられる．Python のように，インデントによるブロック明示を文法に取り入れているプログラミング言語も存在する．タブ（tab）によるインデントと，空白（半角スペース）によるインデントがあるが，使用しているエディタの設定によって何文字分字下げされるかが異なるので注意する必要がある[1]．

- 空白の扱い
  空白（スペース）を適切に入れることにより，コードを見やすくできる．たとえば，変数（定数）と演算子の間，かっこと変数（定数）の間などにスペースを入れることなどが挙げられる．また，数値の場合に，桁の位置合わせにも空白がよく使用される．

[1]半角スペース 4 文字が最もよく採用されている

- ループ制御
  for 文の場合は，ループカウンタの初期化処理，終了判定，更新処理に注意する必要がある．while 文，do-while 文の場合は，ループ開始前のループカウンタの初期化処理，終了判定，ループ内での更新処理に注意する必要がある．for 文でよくある間違いとして，ループカウンタの変数名を初期化，終了判定，更新処理の間で統一していないことなどが挙げられる．while 文の場合，ループカウンタを適切に更新しないことで無限ループに陥ることや，初期化を忘れたり，適切な値を代入しないことによる間違いなどがよく見られるので注意したい．

- 移植性の高いコード（標準化）
  独自の関数を組み込む際に，その機能が標準ライブラリで実装されていたりしないかを必ず確認すること．これは，車輪（部品）の再発明を避けるためである．結果的に，他人がソースコードを読む際に関数の機能や挙動を確認する手間を減らし，可読性を高めることができる．標準ライブラリ関数を使用することによって，コードの移植性を高め，信頼性を高めることにもつながる．

さらに理解を深めたい場合，参考文献 [1], [2], [3], [4] などを参照してほしい．

3 リファクタリング

リファクタリングとは，ソフトウェアの外部的振る舞いを保ったままで，内部構造を改善していく作業のことを指す [5]．リファクタリングでは，コードを統制された方法により洗練（最適化）していくことで，バグの入り込む余地を極限まで減らすことができる．コードを書いてしまった後に，リファクタリングするのが一般的で，設計時には見つからなかった問題点（低可読性や低効率）等を，コーディング（実装）後に改善する．良い設計技法と，プログラミング作法を学んでおくことで，コーディング後のリファクタリングの手間を減らすことも可能となる．

4 ソフトウェア作成のためのツール紹介

ここでは，ソフトウェア作成における便利なツールについて紹介する．

4.1 エディタ

プログラムのソースコードを編集するエディタ（テキストエディタ）は世の中に幾つも存在し，それぞれに特徴がある．ここでは，世界中のプログラマが使用している，言語の種類に依存しない，コーディングに適したテキストエディタを幾つか紹介する．

- Emacs
  非常に多機能なテキストエディタ．ターミナル上 (CUI) でも，GUI でも動作する．作業を自動化するための組込コマンドとマクロを組み合わせることができ，Emacs Lisp を用いることで，Emacs 用の独自のコマンドを作成したり，アプリケーションを作成したりもできる．

- vi (vim)
  Unix 系の OS には標準でインストールされていることが多い，軽快に動作する高機能なテキストエディタ．コマンドモードと挿入モードなど，複数のモードを切替えて使用するのが特

徴的である．主にターミナル上 (CUI) で動作するが，覚えやすいコマンドと，自由にカスタマイズできる点で世界中に多くのユーザが存在する．一方で，日本語などマルチバイト文字での編集には向いていない．

- その他のエディタ

  - gedit
    GUI によるテキストエディタ．GNOME デスクトップの標準的なテキストエディタである．複数のプログラミング言語に対応したハイライト表示やコードスニペットの追加などといったプログラミング向けの機能を持つ．キー操作は Windows 系のショートカットキーである．
  - Atom
    GitHub が開発したオープンソーステキストエディタ．Linux，MacOS, Windows など様々な環境で使用できる．UI は JavaScript や CSS などによりカスタマイズできる．
  - Visual Studio Code
    Microsoft 社製．マルチプラットフォームで動作するソースコードエディタ．C#などの言語のリファクタリング，インテリセンスなどの機能を持つ．

4.2 統合開発環境 (IDE)

IDE（Integrated Development Environment）は，テキストエディタよりも多機能であり，コーディングからデバッグ，ビルド，GUI 設計など，プログラミングに関するすべてのことをおこなえる環境のことを指す．ここでは詳しくは述べないが，主に以下のようなものがある．

- Eclipse
  Java での開発によく使用される．Eclipse 自体が Java で記述されている．
- Xcode
  Apple 社製の統合開発環境．MacOS で使用でき，Objective-C，Swift などの言語での開発に対応している．
- Visual Studio
  Microsoft 社製の統合開発環境．C#，Visual Basic，といった言語での開発ができる．自動補完システム “インテリセンス” が動作する．
- Android Studio
  Android アプリの開発に使用される．Java，Kotlin といった言語に対応している．

4.3 コンパイラ

電算室の環境では，コンパイラとして gcc（GNU Compiler Collection）が使用できる．C 言語以外にも，C++，Objective-C などもコンパイル可能である．コンパイラオプションとして，よく使用するものを以下に挙げる．

- -o **filename**: 実行ファイル名の指定

- -O2: 最適化
- -g: デバッガ利用 (gdb)
- -lm: 数学ライブラリ利用
- -lX11: X Window ライブラリ利用
- -lSDL: SDL のライブラリ利用
- -I**path**: インクルードファイルパス指定
- -L**path**: ライブラリファイルパス指定
- -D**macro-name**: マクロ変数の定義

5 基本テクニック

C 言語によるプログラミングでよく利用されるテクニックを，作成時，デバッグ時それぞれについて記述する．

5.1 作成時のテクニック

5.1.1 命名規則

変数名や関数名には，わかりやすい名前を付けるべきである．とくに，分担作業の場合，他人が利用する可能性のあるグローバルの変数や関数には，誰もが理解しやすい名前を付ける必要がある．ローカルの変数や関数には，当面の間自分だけが読むコードであるため，簡潔かつ意味が推測できる名前が適している．

長すぎる名前や短すぎる名前は，コードを読みにくくするので避けるべきである．達人プログラマのソースコードを読んだり，他人（自分よりもプログラミング経験豊富な人）にコードを評価してもらうことで身につけていくことができる．

1. 変数名には意味を持たせる
   変数名が ‘a’ や ‘b’ などといったただの記号で表されている場合，一見して作者にしか何のための変数かが分かりづらい．変数名を英単語などにすることで，他者が読んだときにも推測可能になる．変数名に，あまり使われない訳語や略語は使用しないほうがよい．一般的な名称を用いること．

2. 関数名には意味を持たせる
   変数名の命名規則と同様，関数名にも，その関数がどんな計算・処理をおこなうかが推測可能な名前をつけるとよい．

3. ファイル名
   プロジェクト内のファイルは，そのファイルに記載される内容によって，一般的な名称が決まっている．

   - README: プロジェクト概要

- INSTALL: インストール手順
- configure: プラットフォーム設定スクリプト
- Makefile: ビルド

4. 拡張子
   標準的な拡張子がある程度決まっているので，それに従うこと．

   - .c: C のソース
   - .h: C/C++のヘッダファイル
   - .cc, .cpp, .cxx: C++のソース
   - .java: Java のソース

5.1.2 関数化

プログラムの見通しをよくするために，main 関数にすべての処理を書くことは避ける．役割ごとに関数に切り分けることでプログラムの修正を容易にする．

5.1.3 マクロ（#define）

#define マクロを使用して定数を定義することで，定数の変更があった場合に，一括で修正可能になる．マクロ名も，命名には注意（何の役割を持つ定数なのかがわかるような名前にする）．

5.2 デバッグ時のテクニック

5.2.1 assert

エラーが起きそうな箇所に仕掛けることで，実行時のエラーの原因特定に役立てる．デバッグが終わったら，NDEBUG を定義してコンパイルすることで，assert() 関数を呼び出さないようにできる．

5.2.2 #ifdef，#ifndef

#ifdef，#ifndef を使用することで，マクロ名が定義されている/されてない場合にその箇所をコンパイルするかどうか指定することが可能になる．これにより，コンパイル時のオプション（-D オプション）を使って複数のバージョンのプログラムを作成することが容易になる．#ifdef，#ifndef を上手く使えば，多重インクルードの防止（インクルードガード）も可能である．

5.2.3 gdb を使用したデバッグ

gdb(GNU デバッガ）を使用することで，実行時のエラー，不具合の原因を探ることができる．gdb を使うためには，コンパイル時に ‘-g’ オプションを付けてデバッグ情報を付与する．また，‘-O0’ オプションにより，最適化しないようにしておく必要がある．デバッグ情報を付与した実行ファイルを gdb 上で実行すると，1 行ごとに実行することができ，どこでどんな問題が起きているのかを変数の内容を見ることで確認したりできる．詳細については，参考文献 [6]，[7] などを参照すること．

6　演習問題

6.1　命名規則について

プログラミングでよく使用される命名規則について調べ，幾つか例を示せ.

6.2　コンパイラオプション

資料に載っていない gcc のコンパイラオプションを調べ，実際の使用例を示せ.

6.3　マクロ

-D オプションを用いて，挙動の異なる二つの実行ファイルを生成するソースコードとそのコンパイル方法について書け.

6.4　assert

assert を使用し，実行時エラーが起きたときに強制終了するプログラムを書け（実際にエラーが起きるように作成すること）.

6.5　gdb

gdb を使用し，前述の assert を使用したプログラムのエラーの原因特定をしてみよ．その際に使用した gdb におけるコマンドについて示せ.

6.6　まとめ

バグの入りにくいプログラムとはどんなプログラムか．200 文字程度で簡潔に述べよ.

参考文献

[1] Diomidis Spinellis（著）：コード・リーディング - オープンソースから学ぶソフトウェア開発技法 - ，毎日コミュニケーションズ，2004.

[2] Diomidis Spinellis（著）：コード・クォリティ - コードリーディングによる非機能特性の識別技法 - ，毎日コミュニケーションズ，2007.

[3] 縣 俊貴，良いコードを書く技術，技術評論社，2011.

[4] 藤原 克則：俺のコードのどこが悪い？，秀和システム，2011.

[5] マーチン・ファウラー（著）：リファクタリング - プログラミングの体質改善テクニック - ，ピアソンエデュケーション, 2007.

[6] リチャード・ストールマン（著），ほか：GDB デバッギング入門，アスキー，1999.

[7] インターフェース編集部： GDB を使った実践的デバッグ手法 - Emacs, Eclipse, Cygwin, Insi - , CQ 出版，2008.