OSコーディングスタイル
～はじめに～

(1)　目的

　この文書は，OS及びアプリケーションを修正及び追加を行う人向け及び,自分がコードを保守しやすくするためのコーディングスタイル文書である.

　コーディングスタイルを作成するにあたって，短い文書とする事を目標とした(以前に具体的な長い文書を作成した時に読まれなかった).

(2)　対象

(品質副特性 JIS X0129-1より抜粋)

・信頼性(成熟性) ソフトのバグの少なさ
・保守性(解析性) コードの理解のしやすさ
・保守性(変更製) コードの修正のしやすさ
・保守性(安定性) 修正による影響の少なさ
・保守性(試験性) 修正したコードのテスト，デバッグのしやすさ
・移植性(環境適応性) 異なる環境への適応のしやすさ

(3)　コーディング作法(コーディング方針)

・定義や処理が明確である事を考慮したコーディング
・誤解を少なくする事を考慮したコーディング
-わずかな資源節約のため，わかりやすさを犠牲にしない
-わずかな速度向上のため，わかりやすさを犠牲にしない
・モジュール指向を考慮したコーディング
・コード再利用のしやすさ(可用性)を考慮したコーディング
・コンパイラやアーキテクチャに依存しないコーディング(アセンブラ，リンカスクリプトを除く)

(4)　前提条件

　前提によって，コーディングスタイルは変わる．前提とは，対象言語と対象ソフトウェア，環境(ツール関係)とする．本コーディングスタイルの前提を以下に示す.

・C言語におけるOSのコーディング
・ツール類を積極的に使用した環境下(過度的なUIは使用しない)

-使用するツールは，make，git，doxygen とする． それ以外のツールは，各個人で使用する．

・ソースコード観覧環境下は，エディタ上

・C 標準ライブラリ関数を使用しない環境下

・信頼性や移植性に関するスタイルよりも，保守性に関するスタイルを多くする

（スタイル 1）　初期化の過不足をなくす
（スタイル 2）　情報損失の危険のある演算は行わない
（スタイル 3）　比較演算の簡略化は行わない
（スタイル 4）　領域 1 つに対して 1 つの目的とする
（スタイル 5）　1 ステート 1 副作用とする
（スタイル 6）　目的の異なる式は分離する
（スタイル 7）　可能なかぎりデータは aggregate でまとめる
（スタイル 8）　リンケージを最小に，モジュール指向プログラミングを行う
（スタイル 9）　1 ファイル内ではコーディングスタイル（括弧，空白，改行，タブ）を統一する（1 ファイルとは，C ファイルとプライベートヘッダファイル）
（スタイル 10）　1 ファイル内では名前の付け方を統一する
（スタイル 11）　1 ファイル内では宣言の記述を統一する
（スタイル 12）　パブリックヘッダファイル内（インターフェース）の記述方式を統一する
（スタイル 13）　/**，/*!，//!，///で始まるコメントは doxygen コメントとし，普通のコメントには使用しない
（スタイル 14）　言語処理系拡張機能及び言語処理系未定義の機能は使用しない
（スタイル 15）　データ型の表現，動作仕様の拡張機能，処理系依存部を確認し文書化する
（スタイル 16）　移植性に問題のあるコードは局所化する
（スタイル 17）　ワーニングレベルを最高にし無視しない
（スタイル 18）　個々の宗教は自分専用のモジュール内のみで唱る
（スタイル 19）　超絶技巧より単純明快を心がける

(スタイル1)　初期化の過不足をなくす
・すべての変数(ローカル変数，static ローカル変数，static グローバル変数，グローバル変数)は初期化してから使用する
・・・等

(スタイル2)　情報損失の危険のある演算は行わない
・暗黙的キャスト
・ビットフィールドに使用する型(unsigned char と signed char)
・・・等

(スタイル3)　比較演算の簡略化は行わない
・NULL チェックの簡略化しない
・真偽値と数値の比較しない
・・・等

(スタイル4)　領域1つに対して1つの目的とする
・変数のリサイクルをしない
・・・等

(スタイル5)　1ステート1副作用とする
・1ステート内での複数関数呼出しをしない
・・・等

(スタイル6)　目的の異なる式は分離する
・ループ初期設定式，継続条件式，再設定式にループ内処理を記述しない
・・・等

(スタイル7)　可能なかぎりデータは aggregate でまとめる
・データ集合体情報をマクロで定義しない(アーキテクチャ非依存部のみ)
・・・等

(スタイル8)　リンケージを最小に，モジュール指向プログラミングを行う
・static，extern で明示する(記憶クラス指定子がない場合は自動的に extern となる)
・グローバル変数の共有より，static グローバル変数+アクセスメソッドのコミュニケーション形式を優先する

・内部ヘッダインクルードガードを行う
・ヘッダファイルは自己完結型を基本とする
・プライベートヘッダファイル(ローカルヘッダファイル)とCファイルを1モジュールとし，パブリックヘッダファイル(グローバルヘッダファイル)はモジュール間インターフェースにする
・プライベートヘッダファイルを異なるモジュールでインクルードしない
・モジュール循環依存(プライベートヘッダ循環インクルード)を避ける
・パブリックヘッダファイルでは，隠蔽のため前方宣言を行う
・モジュールをまたがったグローバル関数(extern関数)は関数マクロ(大文字)でインターフェースの重要性を示す．これは，(スタイル12)にもあてはまる．
・Cにおける2レベルの名前空間をディレクトリとサブディレクトリ，パブリックヘッダファイル，プライベートヘッダファイルと名前規則で補う
・トップディレクトリのヘッダファイルは，C++から呼び出せるようなラッパヘッダファイルとして実装する
・・・等

(スタイル9)　1ファイル内ではコーディングスタイル(括弧，空白，改行，タブ)を統一する(1ファイルとは，Cファイルとプライベートヘッダファイル)

　1ファイル内で，様々なスタイルが混ざっていると，神経に障る事があるので統一する．モジュールとモジュールのスタイルが異なっていても良い．ただし，パブリックヘッダファイル(インターフェース)だけは，スタイル12に従って下さい．．．

　コーディングスタイル(括弧，空白，改行，タブ)に関して，自分の意見を持っている方は，プログラミングに慣れている人である．そのため，新規にモジュールを作成する時は，自分のスタイルを適用して良いが，モジュールに対して修正及び追加を行う時は，そのモジュールで記述されているスタイルに合わせる．その方法が合わない時は，モジュール全体を自分のスタイルで書き直しす(ツールを使えば簡単に可能である)．

(スタイル10)　1ファイル内では名前の付け方を統一する

　自分のスタイルを適用して良い．ただし，シンボルの衝突によるインターポジショニングのエラーを回避できる手段をとらなくてはならない．

(スタイル11)　1ファイル内では宣言の記述を統一する

　自分のスタイルを適用して良い．

(スタイル12)　パブリックヘッダファイル内(インターフェース)の記述方式を統一し，文書化する

パブリックヘッダファイル(インターフェースが記述されたファイル)は，記述方式を統一する．

インターフェースが記述されていないトップディレクトリヘッダファイル(システム内のすべでのモジュールで使用する独自型等の定義がされたヘッダファイル)の記述に従いたくない場合，自分担当のモジュールのプライベートヘッダファイルまたは，個別に変換インタフェースを設けよ．

(スタイル 13)　/**，/*!，//!，///で始まるコメントは doxygen コメントとし，普通のコメントには使用しない

普通のコメントで，/**，/*!，//!，///は使用しない．これらは，本プロジェクトで使用する doxygen コメントである．doxygen コメントは強制ではない．ただし，普通のコメントは記述する事.

doxygen で抽出する内容を以下に示す(強制ではない)．

・ファイルヘッダコメント

```
/*!
  @file ファイル名
  @brief ファイル概要(短く，「XXXX プライベートヘッダファイル」等)
         ファイル詳細説明(2 行目)
  @attention 注意(あれば記述)
  @note メモ(あれば記述)
  @sa 関連情報(あれば記述)
  @date git のコミット番号，変更内容(新規の場合記述しない)
*/
```

・関数ヘッダコメント

```
/*!
  @brief 関数概要(短く，「XXXX する関数」等)
         関数詳細説明(2 行目，あれば記述)
  @attention 注意(あれば記述)
  @note メモ(あれば記述)
  @sa 関連情報(あれば記述)
  @param[in] 仮引数名(型はいらない)
  @param[out] 仮引数名(型はいらない，[out]はポインタの場合)
              @arg 引数がとりうる値域
  @retrun 戻り値の説明(なければ「なし」と記述)
  @retval 戻り値(正常値)
```

  @retval 戻り値(不正値)
  @data gitのコミット番号，変更内容(新規の場合記述しない)
*/

・aggregateコメント
/*!
  @brief 概要
  詳細説明(あれば記述)
  @attention 注意(あれば記述)
  @note メモ(あれば記述)
  @sa 関連情報(あれば記述)
  @data gitのコミット番号，変更内容(新規の場合記述しない)
*/

（スタイル14） 言語処理系拡張機能及び言語処理系未定義の機能は使用しない
・C99(コンパイラによって対応範囲が異なるので，新規格には注意(gccは完全には未対応)
・・・等

(スタイル15) データ型の表現，動作仕様の拡張機能，処理系依存部を確認し文書化する
(例)
・I/O等の固定ビット
・エンディアン
・パディング
・シンボル参照
・パーティション制約(CPU動作モード，メモリ，タイマ)
・メモリマップとメモリセクション情報(bssの初期化等)
・混在言語間インターフェース(Cとアセンブリ)
・unionの扱い(基本的には参照のみとする．スタイルに従いたくない人は，unionの書き込みについて文書化する)
・基本データ型の扱い(特にchar型)

(スタイル16) 移植性に問題のあるコードは局所化する
・処理系拡張マクロ
・アセンブラ
・・・等

(スタイル17)　ワーニングレベルを最高にし無視しない

(スタイル18)　個々の宗教は自分専用のモジュール内のみで唱る

　スタイル9も参照．わかりやすい，見やすい等の自分宗教を仲間に強制させない．例として

・コーディングスタイル(括弧，名前，空白，改行)
・名前規則(～わかりやすい，～見やすい等，ただし，パブリックヘッダファイルは除く)
・ドキュメント管理しない(doxygenを使用しない)コメントの記述
・制御文の扱い(if-ifの連鎖や，switch-caseの有無，goto文，breakやcontinue等)

　また，ツール病(本スタイルで使用を前提としているgit，make，doxygen以外のツール)も移さないようにする．

(スタイル19)　超絶技巧より単純明快を心がける

(超絶技巧の例)

・難解アルゴリズム
・自己書き換えコード(メタ記述)

参考文献

JISX3010 プログラミングC

http://www.jisc.go.jp/

B.W.カーニハン/D.M.リッチー著, プログラム言語C 第2版

Brian W.Kernighan/Rob Pike/福崎 敏博訳, プログラミング作法

MISRA-C研究会, 組込み開発者におくるMISRA-C2004 C言語利用の高信頼化ガイド

Indian Hill Style http://dennou-k.gaia.h.kyoto-u.ac.jp/arch/comptech/cstyle/index.htm

GNUコーディング

http://www.sra.co.jp/public/sra/product/wingnut/standards-j_toc.html

Linuxカーネルコーディング規約

http://linuxjf.sourceforge.jp/JFdocs/kernel-docs-2.6/CodingStyle.html

IPA組込みソフトウェア開発向けコーディング作成ガイド

http://sec.ipa.go.jp/publish/tn06-004.html

福岡知的クラスタ(第Ⅰ期), 組込み現場のCプログラミング標準コーディングガイド

Cコーディングスタイル

http://www.6809.net/tenk/html/prog/cstyl_ex.htm#SEC48

スティーブマコネル/クイーブ, Code Complete<上>-完全なプログラミングを目指して-

ハーブ・サッター/アンドレイ・アレキサンドレスク, C++ Coding Standards