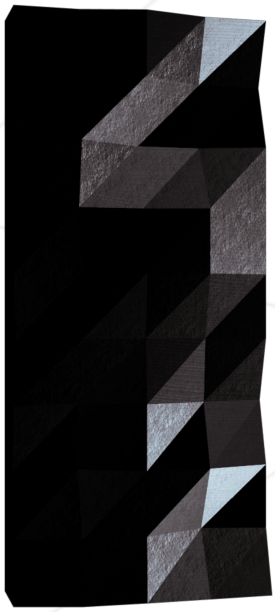

The Norm

Version 2.0.2

Summary: このドキュメントは、*42*で取り扱われている規範（*Norm*）について説明しています。コーディング規範とはコーディングをするときに従わないといけないルールです。明記しない限り、*C*の課題は全て*Norm*に従わなければなりません。

Contents

Chapter I

Foreword

このドキュメントは、42で取り扱われている規範（Norm）について説明しています。コーディング規範とはコーディングをするときに従わないといけないルールです。明記しない限り、Cの課題は全てNormに従わなければなりません。

I.1　なぜ標準に従うの？

Normに従う二つの主な目的： 1、誰も（生徒、スタッフ、それと自分自身）が読んでも理解できるようにコードをフォーマットして標準化するため。 2、あなたのコードが短く簡潔になるようにするため。

I.2　提出時のNorm

あなたのCファイル全てが学校のNormに従わなければなりません。レビュー時に確認されます。規範エラーがある場合は、その問題に対して0点と評価され、その課題自体が 0点と評価されることもあります。ピア・レビュー中は、レビューする側が“Norminette” を提出したリポジトリの中で実行します。“Norminette”はNormの必須の部分だけを確認します。

I.3　提案

あなたはNormが見かけほど威圧的ではないことをすぐに理解するでしょう。それどころか、それはあなたが思っている以上にあなたを助けることでしょう。クラスメートのコードをより簡単に読むことができ、クラスメートも自分のコードを　　簡単に読むことができるでしょう。ソースコードの中にノームエラーが一つあることと10個あることは同じように取り扱われます。コーディング中はNormを意識することを強くお勧めします。初めの頃は自分のコーディング速度が落ちていると感じてしまうかもしれませんが時が経てば反射的にしたがっているはずです。

I.4　免責事項

“Norminette”はプログラムです。そして、全てのプログラムにはバグが存在します。バグを見つけた際には報告してください。“Norminette”は常に正しいべきであるため、報告する際には、相応のバグを報告してください。

Chapter II

The Norm

II.1　名称

必須

- 構造体の名前は必ず`s_`から始まる必要があります。
- typedef（タイプデフ）の名前は必ず`t_`から始まる必要があります。
- 共用体の名前は必ず`u_`から始まる必要があります。
- 列挙体の名前は必ず`e_`から始まる必要があります。
- グローバル変数の名前は必ず`g_`から始まる必要があります。
- 変数と関数の名前は、小文字の英数字と'_'しか含めません(Unix Case)。
- ファイルとディレクトリの名前は、小文字の英数字と'_'しか含めません(Unix Case)。
- ファイルは必ずコンパイルしなければなりません。
- ASCII以外の文字テーブルを使用することは禁止です。

アドバイス

- オブジェクト（変数、関数、マクロ、型、　ファイルそれとディレクトリ）はできるだけ明示的　またはニーモニックな名前を使用する必要があります。カウンタ変数のみ、好きなように名付けてください。
- 略語は、元の名前を短くすること、　およびわかりやすいままである限り許容されます。　もし名前の中に複数の単語がある場合、　'_'で分断してください。
- すべての識別子（関数、マクロ、型、変数）は必ず英語にしてください。
- グローバル変数の使用はすべて　正当である必要があります。

II.2 書式設定

必須

- 全てのファイルは学校のヘッダーをはじめに付け加えなければなりません。このヘッダーはデフォルトで`emacs`と`vim`で利用可能です。
- タブではなく4つのスペースでインデントしてください。
- 各関数は関数の中括弧を除いて最大25行でなければなりません。
- 各行は最大80文字の幅でなければなりません、コメントも同様です。警告：インデントした際も含まれます。
- 1行に1つの命令しか入れられません。
- 空白の行は空白でなければなりません。スペースやタブを入れないでください。
- 行がスペースやタブで終わることはありません。
- 各中括弧の後、または制御構造の終わりに新しい行を開始する必要があります。
- 行末でない限り、各コンマまたはセミコロン後にスペースが必要です。
- 各演算子（2項または3項）または被演算子は1つだけのスペースで区切られます。
- 型（int, char, floatなどなど）やsizeof以外の各C言語のキーワードの後にスペースが必要です。
- 各変数の定義は同じ列でインデントする必要があります。
- ポインタとして表すアステリスクは変数の名前の横につけてください。
- 一行には一つの変数の定義です。
- グローバル変数とスタティック変数以外は変数の定義とその変数の初期化は同じ行では行えません。
- 変数の定義は関数のはじめにあるべきで、その定義の後には一行空けなさい。
- 定義と実装の間には一行あけないでください。
- 多重代入は禁止です。
- 新しい改行を命令か制御構造の後に付け加えることは可能です。しかし、それを行なった場合、インデントと括弧かインデントと代入演算子を入れなければなりません。演算子は必ず、行の初めに入れてください。

II.3 関数のパラメータ

必須

- 関数は、最大4つの名前付きパラメータを入れる事ができます。
- 引数を受け取らない関数は明示的に"void"をパラメータに入れてください。

II.4 関数

必須

- 関数のパラメータは名前付きでなければなりません。
- 各関数の間には一行の空白を入れなければなりません。
- 各関数の中に5つ以上の変数を定義することはできません。
- 関数の戻り値は丸括弧で囲まなければなりません。

アドバイス

- 関数の識別子は同じファイル上の場合、揃える必要があります。ヘッダファイルも同じく揃えてください。

II.5 Typedef, 構造体, 列挙体と共用体

必須

- 構造体、列挙体、共用体を定義する場合、タブを入れてください。
- 構造体、列挙体、共用体を定義する場合、型の所に一つスペースを入れてください。
- typedefの二つのパラメータの間にはタブを入れてください。
- typedefを用いて構造体、列挙体、共用体を定義する場合、これらのルールを沿ってください。 typedefの名前と構造体、列挙体、共用体の名前を揃えてください。
- Cファイルに構造体を定義しないでください。

II.6 ヘッダ

必須

- ヘッダのなかに入れてもいい要素は、ヘッダインクルージョン)、定義、defines、プロトタイプ、マクロです。
- 全てのインクルード(.cか.h)はファイルの先頭にある必要があります。

- 2重インクルードを回避するため、インクルードガードをヘッダに行なってください。ファイル名が`ft_foo.h`の場合、インクルードガードに使用するプリプロセッサのシンボル名は`FT_FOO_H`となります。
- 関数のプロトタイプは.hファイルに明示的に入れなさい。
- 使用しないヘッダインクルージョンは禁止です。

アドバイス

- .cファイルや.hファイルでのヘッダインクルージョンは正当化する必要があります。

II.7 マクロとプリプロセッサ

必須

- 自分で定義したプリプロセッサ定数（もしくは#define）は定数値やリテラルと関連付けるものでなければなりません。
- Normをバイパスするためや、コードを理解させないために定義された全ての#defineは禁止です。これは人間がチェック知らなければなりません。
- 課題が許す限り標準ライブラリに定義されているマクロは使用可能です。
- 複数行にも及ぶマクロは禁止です。
- マクロの名前だけ全て大文字です。
- #if、#ifdefや#ifndefに続く文字はインデントしてください。

II.8 禁止事項

必須

- これらの命令は使用してはいけません。
  - for
  - do...while
  - switch
  - case
  - goto
- 3項演算子の重ね書き
- VLAs - 可変長配列

II.9 コメント

必須

- ソースファイルの中でコメントを残すことは可能です。
- 関数の中にコメントを残すことはできません。
- コメントは改行で始まり、改行で終わります。それ以外の行は、‘**’で初めて全て揃えてください。
- //から始まるコメントは禁止です。

アドバイス

- コメントは英語で書きなさい。有用な情報を残してください。
- あなたのひどいコードで書かれた関数はコメントで正当化されることはありません。

II.10 ファイル

必須

- .cファイルをインクルードはできません。
- .cファイルのなかで五つ以上の関数を定義することはできません。

II.11 Makefile

必須

- $(NAME), clean, fclean, reとallのルールはなければなりません。
- もしMakefileがrelinkした場合、そのプロジェクトは機能していないという判断になります。
- 複数のプログラムを提出しなければならないプロジェクトの場合、上記のルールを含み、それらのプログラムをコンパイルするルールを追加し、それぞれのプログラムをコンパイルするルールも付け加えなさい。
- もしライブラリ(例えば`libft`)に存在する関数をプロジェクトで使用する場合、あなたのMakefileはこのライブラリを自動でコンパイルしなければなりません。
- コンパイルしなければならないソースファイルは全て明示的にMakefileのなかに明記しなさい。