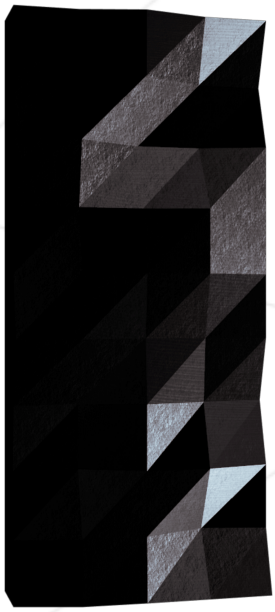

La Norma

Version 2.0.2

Sommario: Questo documento descrive lo standard di scrittura del codice(Norma) di 42. Si tratta di una serie di regole da seguire nella scrittura del codice. Dovrai sempre rispettare la Norma per i progetti in C a meno che non ti venga indicato diversamente.

Indice

Capitolo I

Preambolo

Questo documento descrive lo standard di scrittura del codice(Norma) di 42. Si tratta di una serie di regole da seguire nella scrittura del codice. Dovrai sempre rispettare la Norma per i progetti in C a meno che non ti venga indicato diversamente.

I.1 Perché imporre uno standard?

I due obbiettivi della Norma : 1. Standardizzare il tuo codice in modo tale che risulti di facile lettura a chiunque(studenti, staff ed anche voi stessi). 2. Insegnarti a scrivere un codice corto e semplice.

I.2 La Norma per la consegna

Tutti i tuoi file C devono rispettare la Norma. Sarà la prima cosa che verrà controllata dal tuo valutatore. Se hai anche solo un errore di Norma, otterrai uno 0 per l'esercizio. Durante le revisioni il tuo valutatore userà il comando "Norminette", che controllerà solo la parte obbligatoria della Norma.

I.3 Consigli

Ti renderai ben presto conto che la Norma non è poi così spaventosa. Ti aiuterà più di quello che pensi. Ti permetterà di leggere facilmente il codice dei tuoi compagni di corso e viceversa. Un file sorgente contenente un errore di Norma sarà trattato allo stesso modo di uno che ne contiene 10. Ti consigliamo di tenere a mente la Norma durante la scrittura del codice, anche se inizialmente ti sembrerà rallentarti noterai che col tempo diventerà un riflesso.

I.4 Avvisi

"Norminette" è un programma e, come tutti i programmi, potrebbe essere soggetto a dei bug. Se ti capitasse di incontrarne uno, sei pregato di notificarlo. Ciononostante, dato che "Norminette" ha sempre ragione, tutti i tuoi progetti dovranno adattarsi a questi bug.

Capitolo II

La Norma

II.1 Nomenclatura

Parte Obbligatoriat

- Il nome di una struttura deve iniziare per `s_`.
- Il nome di un typedef deve iniziare per `t_`.
- Il nome di un union deve iniziare per `u_`.
- Il nome di un enum deve iniziare per `e_`.
- Il nome di un global deve iniziare per `g_`.
- I nomi di variabili e funzioni possono contenere solo lettere minuscole, cifre e '_' (Unix Case).
- I nomi di file e cartelle possono contenere solo lettere minuscole, cifre e '_' (Unix Case).
- Il file deve compilare.
- I caratteri che non fanno parte della tabella ascii standard sono vietati.

Parte facoltativa

- Le Entità(variabili, funzioni, macro, tipi file o cartelle) devono avere un nome esplicito o facilmente ricordabile. Per i "contatori" avete potete fare come volete.
- Le abbreviazioni sono tollerate fin quando utilizzate per accorciare il nome originale e rimangono comunque comprensibili. Se il nome contiene più di una parola, devono essere separate da '_'.
- Tutti gli identificatori(funzioni, macro, tipi, variabili, etc) devono essere in Inglese.
- Ogni uso di una variabile globale deve essere giustificato.

II.2 Formattazione

Parte Obbligatoria

- Tutti i tuoi file devono iniziare con l'header standard della scuola. Questo header è disponibile di default con `emacs` e `vim`.
- Devi identare il tuo codice con tabulazioni di 4 spazi. Non sono la stessa cosa di 4 spazi, intendiamo vere tabulazioni.
- Ogni funzione deve avere massimo 25 righe, senza contare le graffe della funzione.
- Ogni riga può avere massimo 80 colonne di larghezza, commenti inclusi. Attenzione: una tabulazione non conta come una colonna, ma quanto gli spazi che rappresenta.
- Una sola istruzione per riga.
- Una riga vuota deve essere vuota: nessuno spazio o tabulazioni.
- Una riga non può terminare con spazi o tabulazioni.
- Devi andare a capo dopo ogni graffa o fine di una struttura di controllo.
- A meno che non sia alla fine di una riga, ogni virgola o punto e virgola deve essere seguito da uno spazio.
- Ogni operatore(binario o ternario) o operando deve essere separato da un solo spazio.
- Ogni parola chiave C deve essere seguita da uno spazio, a parte per quelle che indicano i tipi(int, char, float, etc.) e sizeof.
- Tutte le dichiarazioni di variabili devono essere identate sulla stessa colonna.
- L'asterisco dei puntatori va messo nel nome della variabile.
- Una dichiarazione per riga.
- È vietato dichiarare ed inizializzare una variabile sulla stessa riga. Questo non vale per le variabili globali e per quelle statiche.
- Le dichiarazioni devono essere posizionate all'inizio di una funzione e l'ultima deve essere seguita da una riga vuota.
- Non possono esservi righe vuota tra dichiarazioni o tra implementazioni.
- Le assegnazioni multiple sono vietate.
- Potrai aggiungere una nuova riga dopo un istruzione o una struttura di controllo, ma dovrai aggiungere un identazione con parentesi o operatori di incarico. Gli operatori devono trovarsi all'inizio di una riga.

II.3 Parametri di funzione

Parte obbligatoriat

- Una funzione può avere massimo 4 parametri.
- Una funzione che non prende parametri deve essere prototipata esplicitamente con la parola "void" come argomento.

II.4 Funzioni

Parte Obbligatoria

- I parametri nel prototipo di una funzione devono avere un nome.
- Ogni funzione deve essere separata dalla successiva da una riga vuota.
- Puoi dichiarare solo 5 variabili per blocco.
- Il return di una funzione deve essere tra parentesi.

Parte facoltativa

- Gli identificatori della tua funzione devono essere allineati all'interno dello stesso file. Stessa cosa per i file di header.

II.5 Typedef, struct, enum e union

Parte Obbligatoria

- Aggiungi una tabulazione quando dichiari struct, enum o union.
- Quando dichiari una variabile di tipo struct, enum o union, aggiungi uno spazio nel tipo.
- Aggiungi una tabulazione tra due parametri di un typedef.
- quando dichiari uno struct, union o enum con un typedef, devi applicare tutte le regole. Devi allineare il nome del typedef con il nome dello struct/union/enum.
- Non puoi dichiarare una struttura in file .c.

II.6 Headers

Parte Obbligatoria

- Nei file di header puoi mettere: inclusioni di header(di sistema e non), dichiarazioni, definizioni, prototipi e macro.
- Tutti gli include(.c o.h) devono essere all'inizio del file.

- Devi proteggere ogni file di header dalle doppie inclusioni. Se il file si chiama `ft_foo.h`, la sua macro sarà `FT_FOO_H`.
- I prototipi di funzione dovranno trovarsi esclusivamente nei file .h.
- Le inclusioni di file di header(.h) sono vietate.

Parte consigliatat

- Tutti le inclusioni dei file di header devono essere giustificate sia nei file .c sia nei file .h.

II.7 Macro e pre-processore

Parte obbligatoriat

- Le costanti di preprocessore(o #define) che crei devono essere utilizzate solo per associazioni letterali e valori costanti.
- Tutti i #define creati per bypassare la Norma e/o offuscare il codice sono proibite. Questo punto deve essere verificato da un umano.
- Puoi usare le macro disponibili nelle librerie standard solo se autorizzate nell'ambito del progetto.
- Macro multilinea sono proibite.
- Solo i nomi delle macro devono essere maiuscoli.
- Devi identare i caratteri che seguono #if , #ifdef o #ifndef.

II.8 Roba proibita !

Parte Obbligatoria

- NON puoi usare :
  - for
  - do...while
  - switch
  - case
  - goto
- Operatori ternari annidati tipo '?'.
- VLAs - Array di lunghezza variabile: tab[n].

II.9 Commenti

Parte obbligatoriat

- Puoi commentare il codice nei tuoi file sorgente.
- I commenti non possono trovarsi all'interno del corpo di una funzione.
- I commenti devono iniziare e finire con una riga singola. Tutte le righe intermedie devono iniziare per '**'.
- Niente commenti con //.

Parte facoltativa

- I tuoi commenti devono essere in Inglese ed UTILI.
- Un commento non può giustificare una funzione "illegittima".

II.10 File

Parte obbligatoria

- Non puoi includere un file .c.
- Non puoi avere più di 5 definizioni di funzione in file .c.

II.11 Makefile

Parte Obbligatoriat

- Le regole $(NAME), clean, fclean, re ed all sono obbligatorie.
- Se il makefile ricollega, il progetto viene considerato non funzionante.
- Nel caso di un progetto multibinario dovrai aggiungere, oltre alle regole sopracitate, una regola che compili entrambi i binari così come una regola specifica per ogni binario compilato.
- Nel caso che un progetto chiami una libreria di funzioni (e.g.: `libft`), il tuo makefile deve compilare questa libreria automaticamente.
- Tutti i file sorgente necessari alla compilazione del tuo progetto devono essere citati esplicitamente nel tuo Makefile.