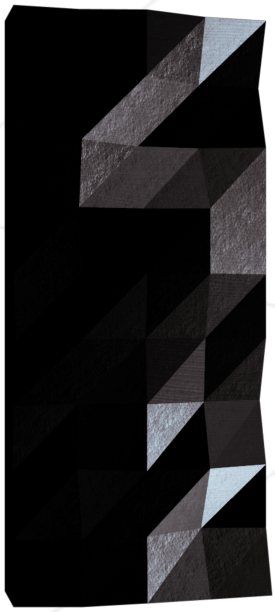

Norm

Sürüm3

Summary: Özet: Bu belge 42'de uygulanabilecek standardı (Norm) tanımlamaktadır. Bir programlama standardı, kod yazarken uyulması gereken kurallar bütününü tanımlar. Norm, aksi kararlaştırılmadıkça Inner Circle kapsamındaki tüm C projelerine, özel olarak belirtildiği takdirde de diğer her türlü projeye uygulanır.

Contents

Chapter I

Önsöz

Norm, Python'da yazılmıştır ve açık kaynaklı bir projedir(?). Veri havuzuna https://github.com/42Scho
üzerinden erişilebilir. Pull request, öneri ve sorunlara ilişkin iletişime geçmekten çekinmeyin!

Chapter II

Norm

II.1 İsimlendirme

- Bir struct'ın ismi mutlaka `s_` ile başlamalıdır.
- Bir typedef'in ismi mutlaka `t_` ile başlamalıdır
- Bir union'ın ismi mutlaka `u_` ile başlamalıdır.
- Bir enum'ın ismi mutlaka `e_` ile başlamalıdır.
- Bir global'ın ismi mutlaka `g_` ile başlamalıdır.
- Değişkenler ile fonksiyonların isimleri yalnızca küçük harfler, rakamlar ve '_' (Unix Case) içerebilir.
- Dosyalar ile dizinlerin isimleri yalnızca küçük harfler, rakamlar ve '_' (Unix Case) içerebilir.
- Standart ASCII tablosunda yer almayan karakterlerin kullanımı yasaktır.
- Değişkenler, fonksiyonlar ve diğer tüm belirleyicilerde snake case kullanılmalıdır. Büyük harf kullanılmamalı ve her bir kelime alt çizgi ile ayrılmalıdır.
- Tüm belirleyiciler (fonksiyonlar, makrolar, tipler, değişkenler vs.) İngilizce olmalıdır.
- Nesneler (değişkenler, fonksiyonlar, makrolar, tipler, dosyalar veya dizinler) mümkün olan en açık ve akılda kalıcı şekilde isimlendirilmelidir.
- Const ve static olarak işaretlenmemiş global değişlenlerin kullanılması yasaktır ve bu durum, projenin bunlara açıkça izin vermediği hallerde norm hatası olarak değerlendirilir
- Bir dosya mutlaka derlenebilmelidir. Derlenemeyen bir dosyanın Norm'a uymasısöz konusu olmayacaktır.

II.2 Format

- Kodunuzu mutlaka 4 boşluk ile indentlemelisiniz. Burada bahsedilen boşluk 4 ortalama boşluk anlamına gelmemekte olup, gerçek anlamda tab tuşuna basılmasını ifade etmektedir.
- Her fonksiyon, fonksiyonun kendi kıvrımlı ayraçları (curly bracket) hariç, maksimum 25 satırdan oluşmalıdır.
- Her satır, yorumlarla birlikte, maksimum 80 sütun genişliğinde olmalıdır. Uyarı: bir kez tablanmış olma bir sütun olarak sayılmamakta, karşılık geldiği boşluk sayısı kadar dikkate alınmaktadır.
- Her fonksiyon yeni bir satır başı ile ayrılmalıdır. Herhangi bir yorum veya ön işlemci, fonksiyonun hemen üzerinde yer alabilir. Satır başı bir önceki fonksiyondan sonra gelir.
- Her satırda tek bir talimat yer almalıdır.
- Boş bir satır mutlaka boş olmalıdır: herhangi bir boşluk veya tab olmamalıdır.
- Bir satır asla boşluk veya tab ile bitemez.
- Hiçbir zaman peş peşe iki boşluk bırakamazsınız.
- Her bir kıvrımlı ayraçtan (curly bracket) sonra veya kontrol yapısının sonunda yeni bir satıra geçmelisiniz.
- Bir satırın sonu olmadığı takdirde, her virgül ve noktalı virgülden sonra bir boşluk bırakılmalıdır.
- Her bir operatör veya operand yalnızca ve yalnızca tek bir boşluk ile ayrılmalıdır.
- Tipler (int, char, float vs. gibi) için olanlar hariç her bir C sözcüğünden (keyword) ve sizeof'tan sonra bir boşluk bırakılmalıdır.
- Her bir değişken declarationı kendi kapsamına göre ilgili sütunda indentlenmiş olmalıdır
- Pointer'larla birlikte kullanılan asteriskler değişken isimlerine bağlı olmalıdır
- Her satırda tek bir değişken declarationı yer almalıdır.
- Global değişkenler (izin verilmesi halinde), statik değişkenler ve sabitler dışında, declarationlar ve initializationlar aynı satırda yer alamaz.
- Declarationlar fonksiyonların başında yer almalıdır.
- Bir fonksiyonda, değişken declarationları ile fonksiyonun geri kalanı arasında boş bir satır bırakmalısınız. Fonksiyonda başka herhangi bir boş satıra izin verilmez.
- Çoklu atamalar kesinlikle yasaktır.

- Bir talimat ya da structuredan sonra yeni bir satır ekleyebilirsiniz, ancak bu durumda ayraçlar (brackets) veya atama operatörü ile bir girinti (indentation) eklemeniz gerekecektir. Operatörler satırın başında olmalıdır.
- Kontrol yapılarında (if, while..), tek bir çizgi ya da tek bir koşul içerdikleri haller dışında, ayraç (brace) bulunmalıdır.

General example:

```
int             g_global;
typedef struct  s_struct
{
    char    *my_string;
    int     i;
}               t_struct;
struct          s_other_struct;

int     main(void)
{
    int     i;
    char    c;

    return (i);
}
```

II.3 Fonksiyonlar

- Bir fonksiyon maksimum 4 isimlendirilmiş parametre alabilir.
- Herhangi bir parametre almamış fonksiyonlar mutlaka parametre kısmına açıkça ‘void ’yazılarak prototiplendirilmelidir.
- Fonksiyonların prototiplerindeki parametreler mutlaka isimlendirilmelidir.
- Her fonksiyon bir sonrakinden mutlaka boş bir satır ile ayrılmalıdır.
- Fonksiyon başına 5 değişkenden fazlasını declare edemezsiniz.
- Bir fonksiyonun geri dönüşü (return) parantez içinde olmalıdır.
- Her fonksiyonun geri dönüş (return) tipi ve ismi arasında tek bir tab bulunmalıdır.

```
int my_func(int arg1, char arg2, char *arg3)
{
    return (my_val);
}

int func2(void)
{
    return ;
}
```

II.4 Typedef, struct, enum ve union

- Bir struct, enum veya union declare edilirken bir tab ekleyin.
- Bir tip struct, enum veya unionının değişkenini declare ederken tipe tek bir boşluk ekleyin.
- Typedef ile bir struct, union veya enum declare ederken, tüm indentleme kuralları uygulanır. Typedef'in adı ile struct/union/enum'ın adını hizalamalısınız.
- Tüm structureların isimlerini kendi kapsamına göre ilgili sütunda indentlemelisiniz.
- Bir structureı bir .c dosyasının içinde declare edemezsiniz.

II.5 Header Dosyaları

- Başlık dosyalarında izin verilenler: başlık inclusionları (sistem veya değil), declarationlar, prototipler ve makrolar.
- Tüm includelar dosyanın başlangıcında olmalıdır
- Bir C dosyası include edemezsiniz.
- Başlık dosyaları çift inclusiondan korunmalıdır. Eğer dosya `ft_foo.h` ise, onun makro karşılığı `FT_FOO_H` şeklindedir.
- Kullanılmayan başlık inclusionları (.h) yasaktır
- Tüm başlık inclusionları hem .c dosyasında hem de .h dosyasında doğrulanmalıdır.

```
#ifndef FT_HEADER_H
# define FT_HEADER_H
# include <stdlib.h>
# include <stdio.h>
# define FOO "bar"

int               g_variable;
struct        s_struct;

#endif
```

II.6 Makrolar ve Ön İşlemciler

- Yarattığınız ön işlemci sabitleri (veya #define) yalnızca gerçek ve sabit değerler için kullanılmalıdır.
- Normu bypass etmek ve/veya kod karıştırmak için yaratılan tüm #define yasaktır. Bu kısım bir insan tarafından kontrol edilmelidir.
- Standart kütüphanelerdeki makroları, yalnızca verilen projelerin kapsamında bunlara izin verilmesi halinde kullanabilirsiniz.
- Multiline makrolar yasaktır.
- Makro isimlerinin tamamı büyük harf olmalıdır.
- #if, #ifdef veya #ifndefden sonra gelen karakterleri indentlemelisiniz.

II.7 Yasaklar!

- Aşağıdakileri kullanma izniniz bulunmamaktadır:
  - for
  - do...while
  - switch
  - case
  - goto
- '? 'gibi ternary operatörleri.
- VLAlar - Variable Length Arrays (Değişken Uzunluklu Dizi)
- Değişlen declarationlarında implicit tip.

```
int main(int argc, char **argv)
{
    int     i;
    char    string[argc]; // This is a VLA

    i = argc > 5 ? 0 : 1 // Ternary
}
```

II.8 Yorumlar

- Yorumlar fonksiyon gövdelerinin içinde olamaz. Yorumlar bir satırın veya kendi satırlarının sonunda olmalıdır.
- Yorumlarınız İngilizce ve kullanışlı olmalıdır.
- Bir yorum 'nahoş' bir fonksiyonu doğrulayamaz.

II.9 Dosyalar

- Bir .c dosyasını include edemezsiniz.
- Bir .c dosyasında 5 taneden fazla fonksiyon tanımına yer veremezsiniz.

II.10 Makefile

Makefilelar Norm tarafından kontrol edilmez, öğrenci tarafından gelişim süresince kontrol edilmelidir.

- $(NAME), clean, fclean, re ve tüm kurallar bağlayıcıdır.
- Makefile relink ederse, proje işlevsiz kabul edilecektir
- Çoklu ikili bir proje söz konusu ise, yukarıdaki kurallara ek olarak, her iki ikiliyi de compile eden bir kural ile compile edilen her bir ikiliye özgü ayrı bir kurala sahip olmalısınız.
- Sistem dışı bir kütüphaneden (libft vs.) fonksiyon çağıran bir proje söz konusu ise, makefileınız bu kütüphane ile otomatik olarak compile etmelidir.
- Projenizi compile etmeniz gereken tüm kaynak dosyalar Makefile'ınızda açıkça isimlendirilmiş olmalıdır.

II.11 Neden?

Norm, pek çok pedagojik ihtiyacı karşılamak amacıyla titizlikle hazırlanmıştır. Yukarıdaki tüm seçenekler için en önemli nedenler şu şekildedir:

- Sıralama: Kodlama büyük ve karmaşık bir görevin uzun bir dizi basit talimatlara bölünmesini sağlar. Tüm bu talimatlar, biri diğerini izleyecek şekilde, sırayla yerine getirilir. Yazılım yaratmaya yeni başlayan bir kişi, projesi için, tüm bireysel talimatlara ve gerçekleştirileceklerin doğru sıralanmasına yönelik tam bir anlayış ile birlikte basit ve yalın bir mimariye gereksinim duyar. Aynı anda birden fazla talimatı yerine getiren kriptik dildeki dizilimler kafa karıştırıcıyken, tek bir kod paydasında karışık bir şekilde yer alan birden fazla göreve işaret eden fonksiyonlar ise hatalara kaynak teşkil eder. Norm sizden, her bir parçanın kendine özgü görevinin açıkça anlaşılabilir ve doğrulanabilir olduğu ve tüm talimatların uygulanmasına ilişkin sıralamanın şüpheye mahal vermeyecek nitelik arz ettiği basit kod paydaları oluşturmanızı talep eder. Fonksiyonlarda maksimum 25 satıra yer verilmesini talep etmemizin ve for, do... while veya ternary kullanımlarının yasak olmasının sebebi de budur.

- Görünüm ve Tavır: Arkadaşlarınızla bilgi alışverişi esnasında veya sizinle aynı pozisyondaki iş arkadaşlarınızla birlikte öğrenme ve ayrıca birbirinizi değerlendirme süreçlerinizde, onların kodlarını deşifre etmekle vakit kaybetmek istemez, doğrudan kodlarının arkasında yatan mantığa dair sohbet etmek istersiniz. Norm sizden, fonksiyon ve değişkenlerin isimlendirilmesi, indent edilmesi, ayraç (brace) kuralları, pek çok yerde kullanılan tab ve boşluklara vs. ilişkin kendinize özgü bir görünüm ve tavır kullanmanızı talep eder. Bu size, başkalarının sizinkilere benzer görünümdeki kodlarını, kodları anlamadan önce okumaya vakit harcamaya gerek kalmaksızın, kolaylıkla incelemeniz imkanını sunar. Norm aynı zamanda bir marka değeri taşımaktadır. 42 topluluğunun bir parçası olarak, iş pazarına girdiğinizde, diğer 42 öğrenci veya mezunları tarafından yazılmış kodları da tanıyor olma imkanına sahip olacaksınız.

- Uzun vadeli vizyon: Anlaşılabilir bir kod yazmak için gerekli çabayı sarf etmek, onun sürdürülebilirliğini sağlamanın en iyi yoludur. Siz dahil herhangi bir kişinin, herhangi bir bugı onarmaya veya yeni bir özellik eklemeye ihtiyaç duyduğu her an, bir önceki kişi işini doğru şekilde yapmış ise, kıymetli vaktinizi kaybetmenizin önüne geçilmiş olacaktır. Bu da kodların zaman kaybı nedeniyle sürdürülebilirliğini yitirmesinin engellenmesini sağlayacak ve pazarda başarılı bir ürüne sahip olmaktan bahsederken fark yaratacaktır. Bunu yapmayı ne kadar erken öğrenirseniz, sizin için o kadar iyi olur.

- Norm'da yer alan kuralların bir kısmının veya tamamının tartışmaya açık olduğunu düşünebilirsiniz, ancak biz ne yapılması ve nasıl yapılması gerektiğine ilişkin çok fazla düşündük ve araştırma yaptık. Sizi fonksiyonların neden kısa olması ve tek bir şey yapmaya yönelik olması gerektiğine, değişkenlerin isimlerinin neden bir anlam ifade etmesi gerektiğine, satırların neden 80 sütun genişliğinden daha uzun olmaması gerektiğine, bir fonksiyonun neden birden fazla parametre almaması gerektiğine, yorumların neden faydalı olması gerektiğine vs. vs. ilişkin bir Google araması yapmaya teşvik etmek isteriz.