C KODLAMA STANDARDLARI

1. KAYNAK PROGRAM KÜTÜKLERI

**1000 satirdan büyük kaynak programlarin hem derlemesi yavas, hem de bakimi zordur.
Programlari 1000 satirlik kütüklere bölün.
79 karakterden büyük satirlar her terminalde ve editörde kolay görüntülenmez. Bu nedenle çok uzun satirlar kullanmayiniz.**

1. KÜTÜK ADI TANIMLAMA KURALLARI.

**Her kaynak program ana kütük adi ve ekten olusur. Ekler genelde derleyici ve kullanilan programa göre düzenlenir (.c, .cc, .l, .y gibi). Ana kütük adi sekiz karakterden olusmalidir.
OKUBENI (README) directory altindaki kütükleri ve derleme adiminda kullanilan parametreleri içermelidir.
make derleme islemi için makefile yerine "Makefile" kullanin.**

2. PROGRAM KÜTÜKLERI.

1. **Her programin basinda kütük içinde ne oldugunu belirten ön bilgi (prologue) olmalidir. Dizi içindeki islevler, tanimlar burada kisaca anlatilir. Gerekirse yazar adi ve yazildigi tarih belirtilir.**
2. **Baslik (header) eklemeleri bu açiklamanin pesine yazilir (include files). Bazi sistemlerde sistemin kullanigi eklemeler, kullanicininkilerin önünde yer almalidir.**
3. **"define" ve "typedef" komutlari bundan sonra yazilir. Önce degismez "macro" tanimlari daha sonra islevsel tanimlar, en son "typedef" ve "enum" tanimlari yapilir.**
4. **Tanimlardan sonra tüm programda kullanilan "global/external" bilgi alanlari tanimlanir. Genel siramada önce "extern", static olamayan global tanimlar, ve sonra static tanimlar yer alir. Bir yapi tanimini ilgilendiren "define" varsa, bu yapi taniminin pesinde yer almalidir.**
5. **Program içinde kullanilan islevler bu tanimlamalarin sonunda yer alir. Belirli bir kural olarak, incelenmesi en kolay olan yönteme göre siralanmalidir. Ayni düzeyde çagirilan islevlerin beraber bulunmasi yararlidir.**

Genel program yapisi :

```
/*
* ön açiklama (ön bilgi)
*/
#include <system_kütükleri.h>
#include <uygulama.h>
#define DEGISMEZLER
#define FUNC(x)
typedef struct A {
...
} a_t;
enum { NO=0, YES};
extern int *p_external;
extern struct A_EXT a_ext;
int *p_global;
struct A_GLOBAL {
...
} a_glob;
#define A_GLOBSZ sizeof(struct A_GLOBAL);
static int *p_static;
main(int argc, char **argv)
{
...
}
```

3. HEADER KÜTÜKLERI

Baslik (header) tanimlari her alt sistem için ayri kütüklerde olmalidir. Makina bagimli tanimlar olasi tasimalarda degistirilmek üzere ayri kütüklerde tanimlanmalidir. Tanimlarda ve eklemelerde (include) kullanilmali "kütükadi" gibi tanimlardan kaçinilmalidir. C derleyicileri -I parametresi ile kütügü nereden alacagini bulabilmektedir. Bu özellik baslik (header) kütüklerinin yerinin degismesi durumunda programlarda degisiklik yapilmasini gerektirmez.
Islevleri ve "external" tanimlari içeren baslik (header) kütükleri tanimin yapildigi kaynak programa eklenmelidir. Böylece derleyici tip denetimini kolaylikla yapabilir. baslik (header) kütükleri iç içe (nested) tanimlanmamalidir. Her baslik (header) kütügündeki ön bilgi alaninada bu baslik (header) kütügünden önce hangilerinin eklenmesi gerektigi anlatilmalidir.

4. DIGER KÜTÜKLER.

OKUBENI (README) adli bir kütügün hem genel görüntüyü tanimlamasi, hem de program derleme ve kullanim biçiminin açiklanmasi açisindan önemi çok büyüktür. Burada kosullu derleme adimlari ve makina bagimli kütükler veya programlar açiklanir.

2. AÇIKLAMALAR HAKKINDA.

Açiklamalar ne oldugunu, nasil yapildigini ve parametrelerin neler oldugunu bildirmelidir. Kisa açiklamalar ise islemin ne oldugunu anlatmalidir. Her islevin basinda 3-10 satirlik bir açiklama her satirda islemin yapilisini ayrintilayan açiklamadan daha iyidir. Blok açiklama

```
/*
*
...
*/
```

biçiminde yazilmalidir. Veri yapilari, algoritmalar blok açiklama içinde anlatilmalidir.

3. TANIMLAR HAKKINDA.

Global tanimlar hemen birinci kolondan baslamalidir. Tüm "external" tanimlarin önünde "extern" bulunmalidir. Eger bir "extern" dizi tanimi (array) varsa bu tanimin boyu her tanimda belirtilmelidir.
Gösterge taniminda kullanilan '*' türün önünde degil, tanimin önünde yer almalidir :

```
char *s, *p;
```

gibi.
Iliskili olmayan tanimlar ayni türden olsalar da ayri satirlarda tanimlanmalidir.
Tanimlarda kullanilan degiskenler, degerler ve açiklamalar alt alta gelecek sekilde "tab" tusu ile ayrilmalidir.
Eger "define" komutundaki degerin program içinde bir anlami yoksa "enum" kullanmak daha iyidir. Örnegin :

```
#define KETCH (1)
#define YAWL (2)
#define SLOOP (3)
#define SQRIG (4)
#define MOTOR (5)
```

yerine :

```
enum bt { KETCH=1, YAWL, SLOOP, SQRIG, MOTOR };
```

Bir degiskenin ilk degeri önemli ise ilk degeri açikça yazilmali, C derleyicisinin degeri belirlemesi beklenmemelidir. "long" olarak tanimlanan degismezlerde "l" yerine "L" kullanilmalidir. Çünkü "2l" ile "21" kolaylikla karisir.
"static" tanimlar mutlaka belirtilmelidir. Hatta STATIC diye bir "define" kullanilmasi daha dogru olur.
Islevlerin geri döndürdügü degerin tipi belirtilmelidir. En çok yapilan hata matemetiksel islevlerin "double" döndürdügünün unutulmasidir.

4. ISLEV TANIMLARI HAKKINDA

Her islevinden önce açiklama alani (prologue) bulunmalidir. Burada islevin ne yaptigi

anlatilmalidir.
Islevin döndürdügü deger mutlaka belirtilmelidir. Eger bir deger döndürmüyorsa "void" tanimlanmalidir.
Islevin her parametresi tanimlanmalidir. Islev içinde kullanilan döngü degiskeni için 'i', karakter göstergeleri (pointer) için 's' ve karakter tanimlamalar için 'c' kullanimi tüm islevlerde ayni amaç için kullanilmalidir. Ayni gruptan olan islevlerde de ayni tür degiskenleri ve parametreleri kullanmak, onlari çagiran programlarda kodlama kolayligi getirir.
Degisken sayida parametresi olan islevlerde C dilinde tanimlanmis "varargs" kullanmak anlasilmasi veya tasinmasi açisindan önemlidir.
Eger islev içinde kullanilan bir degisken kaynak programda tanimli "global" degiskenlerden degil de baska kaynak programda yer aliyorsa "extern" kullanilarak tanimlanmalidir.
Içerlek yazma ve bosluklar islevin blok yapisini gösterir. Her iç blok için en az üç bosluk birakarak yazmak programi daha okunakli yapar.
Uzun kosullarda her ve/veya isleminden sonra kalani baska bir satira yazmak, "for" döngülerinde her bir döngü islemini ayri satira yazmak ve "?" isleminde her bir kosulu ayri satira yazmak programi daha okunakli yapar.

5. BASIT KOMUTLAR (SIMPLE STATEMENTS) HAKKINDA.

Her satirda mümkünse bir islem, komut olsun. "while" döngülerinde döngü gövdesi bos ise ";" ayri bir satirda olsun. "if" deyiminde test sonucunda sifir olmama kosulu derleyicinin kabulüne birakilmasin. Örnegin :

```
if (f() != FAIL)
```

her zaman

```
if (f())
```

biçiminden daha iyidir. Eger FAIL degeri sifir ise ve sonra birisi bu degeri -1 yapmak isterse tüm kodlamada ilgili satirlarin bulunup düzeltilmesi gerekebilir. Bu sekilde kullanim degeri degismese bile diger "if" deyimlerinde de yer almalidir.
Sifir olamayan derleyici kabulü ancak asagidaki testler için kullanilmalidir.
- Sonuç yalniz sifir ve baska bir sey olmuyorsa,
- Sonuç daha önceden adlandirilmis (TRUE gibi) ve baska birsey olmuyorsa, örnegin "isvalid", "valid" veya "checkvalid" gibi islevlerde

kullanilabilir.
Kodlama kolayligi olsa bile birden çok atama ("=" islemi) kullanilmamalidir.
"goto" deyimi hiç kullanilmamalidir. Eger bir döngüden çikmak için gerekiyorsa, döngü içindeki bölüm islev haline getirilmeli, bu islevin döndürdügü deger TRUE/FALSE olarak tanimlanmali ve "goto" deyimi yerine dönen deger kullanilmalidir.
Ancak "goto" deyimi kullanmak gerekiyorsa, etiket (label) programin okunmasina kolaylik saglamasi açisindan, kodlamanin daha solundan yazilmalidir.

6. BIRLESIK KOMUTLAR (COMPOUND STATEMENTS) HAKKINDA.

Kivrimli prantez (brace) içindeki komutlarin tümüne birlesik komutlar denir. Birlesik komutlarda :

```
kontrol {
komut;
komut;
}
```

stili kullanilir. Buna "K & R stili" denir.
"switch" deyimde bir "case" seçeneginden sonra "break" komutu yoksa buraya açiklama içinde bilgi yazin. Eger son seçenek varsayilan (default) degilse mutlaka "break" kullanin ve her zaman son seçenek varsayilan (default) olsun.
"if-else" deyiminde her kosul için komutlar (bir tane de olsa) mutlaka "brace" içine alinsin. Özellikle iç içe tanimlanmis "if" deyimlerinde "else" olmamasi durumunda bu kodlama çikabilecek sorun veya hatayi azaltir.
"do-while" döngülerinde mutlaka "brace" kullanilmalidir.

7. ISLEMLER HAKKINDA.

Tüm ikili islemler ile degiskenler arasinda en az bir bosluk birakin ('.' ve '->' hariç). Eger bir deyimin okunmasi zor ise en az öncelikli islemden deyimi satirlara bölmek gerekir. Gerekli oldugunda parantez kullanarak islem önceliklerini gösterin. Ancak çok fazla parantez kullanmayin. Insan gözü parantezleri okumaya alisik degildir.
virgül (comma) islemi en çok "for" döngülerinde birden çok degiskene ilk deger vermek için yararlidir. Bunun disinda fazla kullanmamaya çalisin.
"?:" islemindeki '?' öncesindeki kosulu parantez içinde yazin.
C dili deyimlerinden "sizeof" disinda kalanlardan sonra ilk parantezden önce bir bosluk birakilmalidir. Islevlerin parameterlerindeki virgülden sonra da bir bosluk birakilmalidir. "macro" tanimlarinda ilk parantezden önce bosluk birakilmamalidir. Yoksa ön derleyici (preprocessor) parametreleri algilayamaz.

8. DEGISKENLERI ADLANDIRMA.

Degiskenin adinin basinda ve sonunda '_' kullanmayin. Bu tür degiskenler kullaniciya açik olmayan derleyici degiskenleri arasinda bulunabilir.
Tüm "define" ve "enum" komutlarinda degismezler için büyük harfli tanim kullanin.
Islev adlari, degisken adlari, "typedef", "struct", "union" ve "enum" tanimlari için küçük harf kullanin.
"macro" islevleri büyük harf olmalidir. Küçük harf "macro" tanimlari eger "macro" bir islev gibi çalisiyorsa kabul edilebilir.
Ayni programda yalniz büyük ve küçük harf farki olan degiskenler ve çok benzer degiskenler kullanmayin (foo ve Foo gibi veya foobar ve foo_bar gibi).
Baska anlama gelebilecek degisken adlari kullanmayin. Mümkünse 'l' harfini hiç kullanmayin. Her zaman '1' ile karisabilir.
"typedef" tanimlarinin sonunda çogu kez "_t" eki bulunur.

9. DEGISMEZLER HAKKINDA

Sayisal degismezler için "define" kullanmak ileride programin bakimini kolaylastirir. Yalniz "define" tanimini degistirmek yeterli olabilir.
Degismezler kullanim amaçlarina uyumlu tanimlanmalidir. Örnegin "long" degismez 'L' ile kayan noktali degismez '.0' ile.
ASCII gösterilemeyen degismezleri "define" altinda tanimlayin veya sekizli (octal) tanimini tirnak içinde kullanin.
NULL yerine '0' kullanmayin.

10. "MACRO" HAKKINDA

Karmasik deyimler "macro" olarak tanimlanir. Eger "macro" parametreleri etrafinda parantezler yoksa islem önceliklerinde sorunlar olabilir.
Bazen hem "macro" hem de islevler ayni adla tanimlanabilir. Bu durumda parametrelerin isleyisi önem kazanir. Islevlerde kullanilan parametrelerin degerleri isleve geçerken, "macro" larda parametrenin açilimi kullanilir.

11. KOSULLU DERLEME

Kosullu derleme islemi makina bagimli islemler, "debug" ve derleme sirasinda belirlenen seçeneklerin kullanimi için önemlidir.
Mümkün ise "ifdef" tanimini baslik (header) kütügüne koyun. Kaynak program içine koymamaga çalisin.

12. DÜZELTME (DEBUG)

"enum" kullanirken mümkünse ilk deger sifirdan farkli olmali. Eger sistemde hata kosulu sifir ise her zaman birinci deger hatayi gösterir.
Her zaman hatayi görmek için eklemesindeki bilgiyi kullanin. Yeri geldiginde "assert" olanagindan yararlanin.

Test amaçli kodlamalarda her zaman baslik (header) içinde "define" ile "macro" kullanin. Böylece kodlamada degisiklik yapmaniz gerekmez.

13. "make" KOMUTU HAKKINDA

"make" komutu için kullanilan bazi genel kavramlar asagida verilmistir :
all Her zaman tüm kütüphaneleri derler.
clean ara kütüklerin tümünü siler.
debug testler için kullanilan 'a.out' uretir.
depend install programlari ve kütüphaneleri gerçek yerine tasir.
deinstall "install" islemini geri alma adimidir.
mkcat yardim ekranlarini "man" komutu ile kullanilir hale getirir.
lint "lint" programini çalistirir.
print/list tüm kaynak programlarin listesini almaya yarar.
rdist kaynak programlari baska bilgisayarlara tasimaya yarar.
Bunlara ek olarak "Makefile" için komut satirindan "DEBUG" veya "CFLAGS" tipi degerler girilebilir.

14. PROJEYE BAGIMLI STANDARDLAR

Genelde bu bölümde proje bagimli kütük adlari, directory adlari, baslik bilgisi adlari gibi tanimlar yazilir.