Ödevlerinizdeki ve labaratuardaki hatalarınızın etkisiyle aşağıdaki konulara değinme ihtiyacı hissediyorum:

- **Genel fonksiyon kullanımı.** Yazdığınız her fonksiyonun düzgün tanımlanmış bir amacı olmalı, fonksiyonun adı bu amaca uygun olmalı ve fonksiyon o amaç dışında bir iş yapmamalı. Genel bir prensip olarak, fonksiyonunuz -özellikle o amaç için yazılmamışsa- kullanıcıyla etkileşime girmemeli. Görevini yerine getirmek için yazdığınız başka fonksiyonları çağırabilir ancak programınızı, birbiriyle alakasız fonksiyonların "benim işim bitti şimdi şu iş yapılmalı o yüzden şu fonksiyonu çağırayım" mantığıyla birbirini tetikleyeceği bir biçimde kurgulamayın. Mesela bir fonksiyonunuz veri üretiyor, bir diğeri bu verileri basıyorsa üreten basanı çağırmamalı, her ikisi de yapılan işi yöneten başka bir fonksiyon tarafından ayrı ayrı çağırılmalı. Fonksiyonlar parçala-yönet mantığının bir uzantısıdır. Bu bağlamda, her fonksiyonun net tanımlanmış bir görevi olmalı, sadece kendi işiyle ilgilenmeli ve programın geri kalanından mümkün olduğunca bağımsız olmalı.
- **Diziler ve boyutları (size - dizinin kullanılan miktarı).** Bir dizi boyut bilgisiyle beraber anlamlıdır. (Stringlerde olduğu gibi) Dizinin özel bir formatı yoksa hiçbir biçimde boyut bilgisi diziden çıkartılamaz; dolayısıyla boyut bilgisi diziyle birlikte saklanır, diziyle birlikte fonksiyonlara gönderilir.
- **Pointer döndüren fonksiyonlar.** Fonksiyonlar çağırıldıklarında lokal değişkenleri için hafızadan yer ayrılır ve fonksiyonun işleyişi bittiğinde bu yer serbest bırakılır. Eğer bir fonksiyon bir lokal değişkenini gösteren bir pointer döndürüyorsa, fonksiyonun işleyişi bittiğinde o pointer artık serbest bir hafıza bölgesindeki rastgele bir değeri gösteriyor olacaktır. Mesela aşağıdaki fonksiyonun çalışması bittiğinde str için ayrılan 10 byte'lık alan serbest bırakılır. Sonradan bu alana başka veriler yazılabilir veya bu alan işletim sistemine iade edilebilir. getName'i çağıran fonksiyon str'nin içeriğine erişmeye çalışırken henüz bu iki olaydan birisi gerçekleşmediyse bir problem çıkmaz, veri yazılmışsa yanlış veri okursunuz, alan işletim sistemine döndürülmüşse segmentation error alırsınız. Böylece bazen düzgün çalışan, bazen lojik hata üreten bazen de çalışma zamanı hatası üreten bir programınız olur. Bu tip hataların (bazen ortaya çıkıp bazen çıkmadıkları için) track ederek bulunması çok zordur.

```
/*An error-prone function that produces a string*/
char* getNameErroneous ()
{
        char str[10]="Evren";
        return str;
}
```

Bu durumu düzeltmenin 2 yolu var:

1. Döndürülecek değişkenin hafıza alanının çağıran fonksiyonda oluşturulup bu alanı gösteren bir pointer'in çıkış parametresi olarak çağırılan fonksiyona verilmesi:

```
char* getName(char* str)
{
        str[0]='E';str[1]='v';...;str[4]='n'; str[5]='\0';
        return str;
}
        …<inside some function>
        char s[10], *name;
        name=getName(s); /*Both name and s show the string*/
```

Bu durumda döndürülen pointer, getName'i çağıran fonksiyon üzerinde tanımlı olan bir diziyi gösterdiği için hafıza bölesi getName'den sonra da rezerve edilir. Stringlerle ilgili labratuvarlarda bunun benzerini yapmıştık.

2. Döndürülen pointer ın heap üzerindeki bir bölgeyi göstermesi:

```
/*Returns a dynamically allocated string which should be freed by the caller
*/
char* getName()
{
        char* str=(char *)malloc(10*sizeof(char));
        str[0]='E';str[1]='v';...;str[4]='n'; str[5]='\0';
        return str;
}
```

Bu durumda döndürülen pointer'ın gösterdiği alanı serbest bırakma sorumluluğu çağıran fonksiyona devredilir ve bu sorumluluk mutlaka comment box'ta ifade edilir. C/C++ gibi dinamik hafıza birimlerinin otomatik olarak serbest bırakılmadığı dillerde bu yöntem hafıza kaçağı hatalarına (ihtiyaç bittiğinde kullanılan hafızanın serbest bırakılmaması) açıktır ve dikkat gerektirir.

- **Diziler ve Pointer'lar.** Bir dizi, kendisine tahsis edilmiş hafıza bölgesinin başını gösteren **sabit** bir pointer ile ifade edilir. Yani diziyi ifade eden değişkenle aynı cinsteki standart pointer arasında 2 temel fark var:
    - Diziye bir hafıza alanı tahsis edilmiştir, dolayısıyla doğrudan veri kaydedilebilir. Standart pointer'a veri kaydetmeden önce onun beli bir hafıza alanını göstermesi sağlanmalı.

```
char str[20], *strP;
strcpy(str, "evren"); /* Good usage*/
strcpy(strP, "evren"); /* Bad usage, no space is allocated for data, produces
logical or runtime error as in the case of getNameErroneous function*/
strP=str;
strcpy(strP, "a new string"); /*Good usage as well*/
```

    - Dizi sabit (const) bir pointer'dır; kendisine tahsis edilen hafıza alanının başı dışında bir yeri göstermesi sağlanamaz.

```
char str[10]="evren", str2[15], * strP;
str2=str1; /*Compile error: str2 is a const pointer, you cannot change where
it shows.*/
str=&str[1]; /*Compile error as well*/
strP= &str[1]; /*Good Usage*/
```

Bunun dışında dizi ve pointer'lar benzer sentaks kurallarıyla kullanılabilir. Aşağıdaki kullanımlar, "s"'in bir dizi olarak tanımlandığı durumda da bir diziyi gösteren bir pointer olarak tanımlandığı durumda da geçerlidir:

```
s[1], *(s+1) : 2nd element
&s[1], (s+1) : Adress of the second element
```