Ders 1

Bölüm 1: C'ye Giriş

C'ye Giriş... 'e Giriş... Dur, ne?

C diline giriş yaparken şu 4 noktaya değinerek başlıyorum:

1. Dilin tarihi ile ilgili bir işimiz olmadığı için tarihine değinmeyeceğiz.
2. Soru sormaktan ve deney yapmaktan **çekinmeyin**. İster bana isterseniz birbirinize sorular sorabilirsiniz.
3. CS Komünitesinin hazırladığı derslere de göz atabilirsiniz. Ders anlatımlarımız farklı olabilir, belki bende anlamadığınız bir şeyi orada anlayabilirsiniz.
4. Yazdığımız kodun bir **programa** dönüşmesi için bir yorumlayıcıya ihtiyacımız var.
   Bunlar yorum parçaları hariç bütün kodu işleyerek çalıştırabileceğimiz bir program yaparlar.

Yorumlar

2 tür yorumumuz var:

1. Satır sonu yorumları:
   `//` ile başlar, satırın sonuna kadar devam eder.
2. Çok satırlı yorumlar veya ara satır yorumları:
   `/*` ile başlayıp `*/` ile biterler.
   Aynı satır üzerinde bitmek zorunda olmadıkları için birkaç satırlık yorum oluşturmada kullanılırlar.

Selam C! Naber?

Şu koda bi göz atalım:

```c
#include <stdio.h>
//Kullanıcıyı adıyla selamlayalım!
int main(int argc, char* argv[]){
    char* name;

    if(argv[1] != NULL){
        name = argv[1];
    } else {
        char tmp_name[51];
        printf("Ismin nedir?: ");
        scanf("%50s",tmp_name);
        name = tmp_name;
        printf("\n\n");
    }

    printf("Selam, %s!\n",name);
    return 0;
}
```

Biraz karışık gözüküyor, hadi programı nereye nasıl yazdığımıza ve değişkenlere kısaca baktıktan sonra girdi/çıktıdan başlayalım!
Veri tiplerini birazdan anlatacağım.

Programı nasıl yazarım?

İndirip kurduğumuz **Dev-C++** programıyla. Eğer indirmediyseniz lütfen indirilecekler pdf'ine bakarak indirin ve kurun.
Dev-C++ ile yeni bir dosya oluşturalım ve onun içine kodlarımızı yazalım.

Programı nereye yazarım?

Basitçe, başlangıçta yazacağımız bütün kodlar şu iki kodun arasında sandiviç olacak:

Sandiviçin üstü:

```
#include <stdio.h>
int main(){
```

Sandiviçin tabanı:

```
}
```

İşte tam bu iki kod parçası arasına kodumuzu yazıcaz, sandiviç yapacağız, ve program yorumlayıcısına servis edeceğiz.

Değişkenler... Matematik?

Programlamadaki ve matematikte değişkenler farklı şeyler, içiniz rahat olsun, sadece bir noktası aynı:
Değişkenlerin değeri değişebiliyor, bu matematikte nasıl *hemen hemen her değer* için formül yazmamızı sağlıyorsa, program yazarken de değişkenler sayesinde bir çok değeri tek kod ile kullanabiliyoruz!
Teorik olarak şunları belirtip girdi ve çıktılara girelim:

1. Her değişken bellekte yer tutar, ve tuttuğu yerin **adresi** vardır.
   Ayrıca, modern bilgisayarlardaki bellek bir çok program tarafından kullanılıp bırakıldığından, değişkene atama yapmadığınız zaman garbage data - yani çöp veri - bulundururlar.
   Kullanacağınız çoğu editör bu konuda sizi uyarır, uyarmasa bile bunu alışkanlık haline getirmek sizin yararınıza olur.
2. Değişkenlerin tipi vardır, bu tipler bellekte tuttuğu yer, aritmetik işlem metodu gibi bir çok şeyi etkileyebiliyor.
   C, strong-typed - yani sert dilli - bir dil olduğu için tip konusunda biraz daha sert davranır.
   Javascript gibi loose-typed - yani gevşek dilli - dillerde tip zorlamaları program çalıştırıcısında hallediliyor ve de bu konuda daha gevşek.

Girdi! Çıktı!

Girdi ve çıktı, diğer bir değişle: programın her koşula* ayak uydurabilmesi(girdi) ve programın çalıştığını anlayabilme veya programın meyvesini yeme(çıktı).

Girdi ve Çıktı konuları aslında çok genel konular, hadi bunlar için basit örnekler verelim, nelerin girdi veya çıktı olduğu hakkında bi fikrimiz olsun:

- Bir web sunucusu!
  Girdisi: Bir gönderi (veya "durum") oluşturduğunuzda gönderdiğiniz metin, giriş yapmak için kullandığınız veri, veya en basiti: web adresi.
  Evet, web adresi de bir girdi sayılabilir çünkü sunucu her adres için farklı bir sayfa gönderebilir.

Çıktısı: Ham veriler, mesela web sayfasının kendisi, bir yayın (online TV veya radyolar) veya başarısızlık durumu (404)!

- Telefonunuz!
  Girdisi say say bitmez aslında: Ekranın dokunmatik kısmı, tuşları, içindeki sensörleri, mikrofonu, kamerası, vs, vs...
  Çıktılar için de onu diyebiliriz: Ekranın monitör kısmı (yani görebildiğimiz kısmı), hoparlörleri, titreşim motoru, üzerinde özel amaçlar için kullanılan LEDler veya flaş ışığı.
- E bizim program?
  Girdileri: Terminal içinde ya klavyemizden metin ile ya da argüman şeklinde (tartışmıyoruz, merak etmeyin.)
  (Yok aslında merak edin ya, 3. Bölüm'de argüman kimdir kimdendir anlatıcağım.)
  Çıktısı: Terminale yazı yazdırıyoruz.

printf ve ilk programın ilk meyvesi

`printf` kimdir? printf, print ve format kelimelirinin birleşimi ile adlandırılmış bir fonksiyondur. Özetle, verdiğimiz şekil(format) ile ekrana/terminale yazı yazdırır(print).

Örnekler!

1. Sadece verdiğimiz yazıyı yazsın:
   Kod:

```
printf("Selam Selami!");
```

   Çıktı:

```
Selam Selami!
```

2. Verdiğimiz bir sayıyı yazsın:
   Kod:

```
printf("Şanslı sayınız: %d",rastgele_sayi_yap());
//rastgele_sayi_yap fonksiyonu rastgele bir sayı döndüren hayali bir fonksiyondur.
```

   Çıktı:

```
Şanslı sayınız: 3
```

3. Among Us oyunundaki tanıdık bir metin ☺:
   Kod:

```
printf("%s was the Impostor.", "Red");
```

   Çıktı:

```
Red was the Impostor.
```

Eee... bu yüzdeliler ne? Şekil diyodun?

Format/Şekil metinlerine printf'in anatomisiyle birlikte girelim:

```
printf( char* format_metni, ...degerler );
```

- `format_metni`: Bu metnin içinde % ile başlayan özel belirteçler, sağdaki değerleri kullanarak dinamik metin yazdırmanızı sağlıyor.
  Yani diğer bir deyişle bu arkadaş sayesinde kırk farklı fonksiyon kullanmadan çok çok değerli metinleri çıktı verebiliyoruz. Sevin sevidirin.
- `...degerler`: Başında `...` olan argümanlar 0 veya daha fazla argümanı temsil ediyor, bunu önden açıklamış olalım.
  Şimdi, bu değerler neye göre geliyor?
  Değerler, format metnindeki özel belirteçlere göre (hani şu % ile başlayanlar) geliyor.
  Hadi bunların başlıcalarına bakalım:
  - `%d`: Tamsayı yazdırır.
  - `%s`: Bir metin yazdırır.
  - `%c`: Bir karakter yazdırır.
  - `%f`: Kayan noktalı sayı yazdırır.

scanf ile girdi, programın dinamikleşmesi

scanf, yani scan format fonksiyonu, terminal ekranından istediğimiz tipte veri almamızı sağlıyor. Bu aldığımız verileri kaydetmek için bir değişkene ihtiyacımız var, değişkenin yerini söylemek için de scanf'e değişken **adresi** veriyoruz.

Bir örnek olarak, kullanıcıdan bir zar için yüz sayısı isteyelim, önce sorumuzu soralım:

```
printf("Zarın kaç yüzü olsun?: ");
```

Sonra da değeri alalım:

```
int yuz_sayisi;
scanf("%d", &yuz_sayisi);
```

Basit, değil mi?

Anatomisine de bakacak olursak:

```
scanf(char* format_metni, ...&degiskenler)
```

scanf'in anatomisi printf'e oldukça yakın, bu yüzden sadece farklı olan kısımlarına değinelim:

1. printf'e değer, scanf'e değişken adresi veriyoruz:
   printf sadece dışarıya bir şeyler yazmak istiyor bu yüzden fonksiyonu çağırırken değişikenin değer kısmı yetiyor.
   Ama scanf kullanıcıdan aldığı değeri *kaydedeceği için*, aldığı değeri **nereye** yazacağını bilmesi gerekiyor.
   Bunun için de scanf'e **adres** gönderiyoruz.
2. `%s` biraz daha farklı çalışıyor:
   Metin toplarken scanf, sonraki boşluk-yapacak karaktere kadar arıyor. Bu boşluk-yapacak karakterlere boşluk, Tab, hatta satır başı (newline) örnek verilebilir.

Ayrıca, `%s` kullanırken metnin toplanacağı yerin bir daha adresini almıyorsunuz, C'de metinler zaten bir adres.
Adrese Bağlı Veri Tipleri'ni açıklarken bu konuya değineceğiz.

3. Format metninde boşluk da farkı çalışıyor:
   printf'te bir tane boşluk koyduğumuzda bir tane boşluk koyuyordu.
   scanf'te bir tane boşluk koyduğumuzda sonraki boşluk-yapmayacak karaktere atlayarak devam ediyor.
   Yani, `asd    213` ve `asd 213` scanf'in gözünde aynı. Hatta

   ```
   asd
   213
   ```

   bile scanf için yukardakiler ile aynı.

Temel girdi ve çıktımızı halletiğimize göre, veri tiplerini öğrenelim.

Veri Tipleri

Değişkenler kısmında değişkenlerin hafızada yer kapladığını, her birinin hafıza adresi olduğunu ve tipe göre bazı farklılıklar gösterdiklerini söylemiştik. Hadi bu tipleri tanıyalım, sonra da adres ve adrese bağlı tipleri tanıyalım.

- Tamsayılar: `int, short, long, long long`
  Tamsayıları temsil eden değişkenlerdir.

  ```
  int onluk = 255;
  int onaltilik = 0xff;
  int sekizlik = 0377;
  int ikilik = 0b11111111
  ```

  En yaygın gösterimi sayıyı olduğu gibi yazmak, ama başka şekilde de yazabiliyoruz:

  - Onluk sistemde (Decimal): `255`
    Başında 0 **olmamasına** dikkat edin.
  - Onaltılık sistemde (Hexadecimal): `0xff`
    Başındaki `0x`'e dikkat edin: bu önek sayesinde sağındaki kısmı sayesinde sağ tarafı onaltılık sistemde alıyor.
    Onaltılık sistemin ilk 10 sembolü 0'dan 9'a, diğer 6 sembolü de a'dan f'ye karakterlerdir. a'dan f'ye olan kısmın büyük veya küçük olması farketmiyor.
  - Sekizlik sistemde (Octal): `0377`
    Sekizlik sistemin öneki `0`, bu biraz kafa karıştırıcı olabiliyor, bazı dillerde de `0` yerine diğer öneklerle uyumlu olması için `0o` kullanılıyor. Malesef C'de bu sadece `0`.
    Sekizlik sistemin sembolleri 0'dan 7'ye kadar. İlk 8 doğal sayı olarak aklınızda kalabilir.
  - İkilik sistemde (Binary): `0b11111111`
    Yazması en uzunu bu, ama aynı zamanda bazı bitlik işlemleri yaparken matematik kullanmamanızı da sağlıyor, o yüzden bunu da sevin, sevidirin.
    Öneki `0b`.
    İki tane sembolü var ve elektrik şalterlerinde de görmüş olabilirsiniz. "Kapalı" olan 0 ve "açık" olan 1.

  İşlemler olarak aritmatik temel işlemlerin hepsini destekliyor.

  - Toplama: `2 + 2`
  - Çıkarma: `4 - 1`

- Çarpma: `4 * 4`
- Bölme: `64 / 8`
- Kalan: `20 % 3`
- Karşılaştırmalar: `2 < 3`, `3 == 3` `3 <= 3`, `4 > 2`, `4 >=3` ve `4 != 1`

Tip kelimesi olarak `short`, `int`, `long` ve `long long` aralarındaki tek fark hafızada kapladığı yerdir.
Sisteme ve compiler'a göre değişeceği için tam bir değer söyleyemeyiz ama boyutlarını karşılaştırırsak `short <= int <= long <= long long` diyebiliriz.
Ayrıca, `int` kendi başına kullanıldığında negatif değer alabilir, yani işaretlidir (signed).
Negatif sayıları o değişkende kullanmayacaksak başına `unsigned` yani işaretsiz yazarak pozitif sayılardaki kapasitemizi iki katına çıkarabiliriz.
Mesela `int` 8 bit kaplıyorsa, normal aralığı [-128, 127], işaretsiz aralığı [0, 255] olacak.

- Kayan noktalı sayılar -- floating point numbers `float, double, long double`

```
double duz = 3;
long double ayrica_duz = 3l;
float pi = 3.14f;
long double buyuk = 1e100l;
long double minik = 1e-100l;
```

Kayan noktalı sayılar çok büyük ve çok küçük sayıları göstermek için kullanılırlar.
Buradaki çok büyük ve çok küçükten kast ettiğim + veya - olması değil, `a * 10^x` tanımındaki x'in çok büyük veya küçük olması. Yani bir kuadrilyonu göstermek için de, nanometrenin metreye oranını göstermek için de kayan noktalı sayılar kullanırız.
Tamsayılar gibi bunlar da aritmatik işlemleri destekler.
Tip isimleri yine değişkenin hafızadaki boyutunu değiştiriyor, bu da aslında değişkenin **hassasiyetini** etkiliyor.
Kapladığı alan olarak yine `float <= double <= long double` diyebiliriz.

Kayan noktalı sayıların ayrıca birkaç özel değerleri var.

- NaN: Not a Number, yani bir sayı değil'in kısaltması.
  NaN'ların özel koşulu karşılaştırıldıklarında sürekli `false` değerini döndürürler. Buna kendileri de dahil.
- Infinity ve -Infinity: Sonsuz konseptini temsil ederler. Değer olarak en büyük ve en küçük `float` sayılardır.

- Karakter: `char`
  En küçük alan kaplayan tipimizdir.
  Her ne kadar bir karakteri temsil etse de sayısal değer de saklar.

```
char A = 'A';
char A_sayisal = 65;
char yeni_satir = '\n';
```

Sayısal gösterimler karakterlerin ASCII koduna denk geliyor.

Ascii karakterlerin tablosunu internette bulabilirsiniz.

Buradan sonrası Ders 2'de.