



RUST UYGULAMA Mini GREP Command Line Aracı

**(komut satırını kullanarak metin dosyasında
arama yapmak)**



Programın Amacı

Komut satırından kullanılacak

Komut satırında iki tane argüman verilecek

Birinci argüman "arancak metin"

İkinci argüman "içinde arama yapılacak metin dosyasının adı/yolu"

cargo run -- aranacak_metin arama_yapılacak_dosya (cargo run dan sonra iki adet tire işareti var)

Son olarak, aranan metnin bulunduğu satırlar bir çıktı dosyasına yazdırılacak.

cargo run -- aranacak_metin arama_yapılacak_dosya > output.txt



Program ile Öğrenilecekler

Komut satırından argüman almak ve işlemek

Kod refactor - (program kodlarını birbiri ile olan bağına göre ayrıştırmak - kod tekrarını engellemek - programı geliştirmeyi kolaylaştırmak)

Modüler system

Metin dosyasını okumak

Hata yönetimi

Kullanıcı tanımlı veri yapısı kullanımı

Closures

Iterators

Programı test etmek.

Ortam değişkenlerinin kullanımını



Komut satırından gelen argümanları okumak:
std::env::args;

args() → iteratör döndürür.

args().collect() → Vectör döndürür

```
fn main() {  
  
    let gelen_veriler: Vec<String> = env::args().collect();  
    dbg!(args);  
}
```

cargo run → Bize proje yolunu verir.

cargo run - arancacakdeğer dosyaadı → Üç tane değer verir.

dbg!→stderr, ifadenin sahipliğini alır ve döndürür

println!→ stdout, referans alır



Gelen argümanları değişkende saklamak :

```
let aranacak_deger = &gelen_veriler[1];  
let dosya_yolu = &gelen_veriler[2];  
  
println!("Aranacak metin : {}", aranacak_deger);  
println!("Metnin aranacağı dosya : {}", dosya_yolu);
```




Metin dosyasından verileri okumak :

```
let dosya_icerigi = fs::read_to_string(dosya_yolu).expect("Dosya okunamadı");  
println!("With text:\n{dosya_icerigi}");  
( main.rs )
```



Code Refactor

Projeye modüler yapı kazandırmak :

- Komut satırından gelen veriler
- Program içindeki veriler
- Hata yönetimi

şeklinde gruplandıracağız.

Komut satırından gelen argümanlar ile program içinde kullanılan değişkenler birbirinden farklı yapılardır.

"aranacak_deger" ile "dosya_yolu" programımız için ayar değişkenleri olarak düşünebiliriz. (Yapılan işlem kısa olduğu için main.rs te tutabiliriz. Kodlar artarsa lib.rs ye taşıyabiliriz.)

"dosya_içerigi" ise programımızın yapacağı iş (logic) ile ilgili kısımdır. O yüzden bununla ilgili kodları lib.rs dosyasına taşıyacağız.



main.rs'nin görevleri:

- komut satırı parsing işlemi
- varsa diğer ayar bilgilerini almak
- lib.rs'den fonksiyon çalıştırmak
- lib.rs'deki fonksiyondan gelen hatayı işlemek.

lib.rs görevleri :

- arama işi ile ilgili bütün kodlar burada yer alacak.

```
let (aranacak_deger, dosya_yolu) = komut_satiri_arg_al(&gelen_veriler);  
(main.rs)
```

```
( lib.rs )  
fn komut_satiri_arg_al(gelen_veriler: &[String]) -> (&str, &str) {  
    let aranacak_metin = &gelen_veriler[1];  
    let dosya_yolu = &gelen_veriler[2];  
  
    (aranacak_metin, dosya_yolu)  
}
```




Komut Satırı Argümanlarını Biraraya Toplama

```
pub struct Ayar {  
    pub aranacak_deger: String,  
    pub dosya_yolu: String,  
}
```

```
fn komut_satiri_arg_al(gelen_veriler: &[String]) -> Ayar {  
    let aranacak_deger = gelen_veriler[1].clone();  
    let dosya_yolu = gelen_veriler[2].clone();  
  
    Ayar {  
        aranacak_deger,  
        dosya_yolu,  
    }  
}
```



Metod Kullanarak Ayar Örneği (instance) Oluşturmak

```
impl Ayar {  
  
    fn new(gelen_parametreler: &[String]) -> Ayar {  
        let aranacak_deger = gelen_parametreler[1].clone();  
        let dosya_yolu = gelen_parametreler[2].clone();  
  
        Ayar {  
            aranacak_deger,  
            dosya_yolu,  
        }  
    }  
}
```

Kullanıcının yanlış sayıda parametre göndermesine karşılık önlem almak



```
fn new(gelen_parametreler: &[String]) -> Ayar {  
    if gelen_parametreler.len() < 3 {  
        panic!("Gerekli parametreler verilmedi...!");  
    }  
}
```

```
let ayar = Ayar::new(&gelen_veriler);
```



```
pub fn insa_et(gelen_parametreler: &[String]) -> Result<Ayar, &'static str> {  
    if gelen_parametreler.len() < 3 {  
        return Err("Gerekli parametreler verilmedi...!");  
    }  
    let aranacak_deger = gelen_parametreler[1].clone();  
    let dosya_yolu = gelen_parametreler[2].clone();  
  
    Ok(Ayar {  
        aranacak_deger,  
        dosya_yolu,  
    })  
}
```

Hatanın İşlenmesi (clousers ile)



```
let ayar = Ayar::insa_et(&gelen_veriler).unwrap_or_else(|hata| {  
    println!("Argumanları alırken hata oluştu..! Hata Mesajı : --- {hata}--");  
    process::exit(1);  
});
```




Logic kısmın main içinden çıkartılması (dosya okuma işlemini de lib.rs ye taşıyacağız)

```
fn calistir(ayar: Ayar) {  
    let dosya_icerigi = fs::read_to_string(ayar.dosya_yolu).expect("Dosya  
okunamadı...!");  
    println!("Dosya içeriği : \n {dosya_icerigi}");  
}
```

(lib.rs)

```
    calistir(ayar);  
( main.rs )
```



calistir() fonkisiyonuna hata yönetiminin eklenmesi

```
pub fn calistir(ayar: Ayar) -> Result<(), Box<dyn Error>> {  
    let dosya_icerigi = fs::read_to_string(ayar.dosya_yolu)?;  
  
    println!("Dosya içeriği : \n {dosya_icerigi}");  
  
    Ok(())  
}
```



Dönen hatanın main.rs de işlenmesi

```
if let Err(e) = calistir(ayar) {  
    println!("Uygulama hatası oluştu : -- {e} --");  
    process::exit(1);  
}
```



ARAMA YAPAN LOGIC KISMININ YAZILMASI

TEST YAZMAK :

lib.rs ve main.rs'den bütün println satırlarını pasif hale getirin.

Aşağıdaki basit bir test işlemidir.



```
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn one_result() {
        let aranan_deger = "duct";
        let dosya_icerigi = "\
```

```
Rust:
safe, fast, productive.
Pick three.";
```

```
    assert_eq!(
        vec!["safe, fast, productive."],
        metin_ara(aranan_deger, dosya_icerigi)
    );
}
```

```
}
```

Metin içerisinde "duct" kelimesini aramaktadır.

Başarısız olan bir testtir.

Çalışması için metin_ara() fonksiyonunu eklememiz gerekir.

Testin başarısız olması için fonksiyon kodları yazılmamıştır.

```
pub fn metin_ara<'a>(aranan_deger: &str, dosya_icerigi: &'a str) -> Vec<&'a str> {
    vec![]
}
```




Testi çalıştırmak için;

cargo test

Başarılı olması için; (ARAMA FONKSİYON KODLARININ TAMAMLANMASI)

```
pub fn metin_ara<'a>(aranan_deger: &str, dosya_icerigi: &'a str) -> Vec<&'a str> {  
    let mut bulunanlar = Vec::new();  
    for satir in dosya_icerigi.lines() {  
        if satir.contains(aranan_deger) {  
            bulunanlar.push(satir);  
        }  
    }  
    bulunanlar  
}
```



metin_ara() fonksiyonunu çalıştır() fonksiyonunda kullanarak programımıza işlevsellik kazandıralım

```
pub fn calistir(ayar: Ayar) -> Result<(), Box<dyn Error>> {  
    let dosya_icerigi = fs::read_to_string(ayar.dosya_yolu)?;  
  
    for satir in metin_ara(&ayar.aranacak_deger, &dosya_icerigi) {  
        println!("{satir}");  
    }  
  
    Ok(())  
}
```

Programımız artık arama işlemini yapacaktır. Aranılan kelimelerin geçtiği satırlar ekrana yazılır. Aranılan değer yoksa çıktı olmayacaktır.

```
cargo run -- rust metin_dosyası_adi.txt
```



Ortam Değişkenleri İle Büyük/Küçük Harfe Duyarlı İşlevsellik Ekleme

- Kullanıcının tanımlayacağı ortam değişkenine göre büyük/küçük harfe duyarlı olmayan bir arama seçeneği ekleyeceğiz.
- (Bu işlem komut satırına argüman eklenerekte yapılabilir tabiki de)
- Ortam değişkeni ayarlandığında aramalarda case sensitive kullanılmayacak.

```
pub fn metin_ara_case_insensitive<'a>(aranan_deger: &str, dosya_icerigi: &'a str) ->
Vec<&'a str> {
    let aranan_deger = aranan_deger.to_lowercase();
    let mut bulunanlar = Vec::new();

    for satir in dosya_icerigi.lines() {
        if satir.to_lowercase().contains(&aranan_deger) {
            bulunanlar.push(satir);
        }
    }

    bulunanlar
}
```



Başarısızlık Testi Ekleme :

```
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn case_sensitive() {
        let query = "duct";
        let contents = "\
```

Rust:
safe, fast, productive.
Pick three.
Duct tape.";

```
        assert_eq!(vec!["safe, fast,
productive."],
metin_ara_case_insensitive(query,
contents));
    }
```

```
#[test]
fn case_insensitive() {
    let query = "rUsT";
    let contents = "\
```

Rust:
safe, fast, productive.
Pick three.
Trust me.";

```
        assert_eq!(
            vec!["Rust:", "Trust
me."],
            metin_ara_case_insensitive
(query, contents)
        );
    }
}
```

Testin başarısız olması için `metin_ara_case_insensitive()` içini boş bırak, sadece `vec![]` döndürülsün.

Ortam Değişkeni İçin Ayar Yapısına Alan Eklemek:



```
pub struct Ayar {  
    pub aranacak_deger: String,  
    pub dosya_yolu: String,  
    pub b_k_harf_duyarlı: bool,  
}
```




çalıştır() fonksiyonu (henüz çalışmayacak)

```
pub fn calistir(ayar: Ayar) -> Result<(), Box<dyn Error>> {  
    let dosya_icerigi = fs::read_to_string(ayar.dosya_yolu)?;  
  
    let bulunanlar = if ayar.b_k_harf_duyarli {  
        metin_ara_case_insensitive(&ayar.aranacak_deger, &dosya_icerigi)  
    } else {  
        metin_ara(&ayar.aranacak_deger, &dosya_icerigi)  
    };  
  
    for satir in bulunanlar {  
        println!("{satir}");  
    }  
  
    Ok(())  
}
```



Ortam Değişkeninin Kontrol Edilmesi

std::env;

```
pub fn insa_et(gelen_parametreler: &[String]) -> Result<Ayar, &'static str> {  
    if gelen_parametreler.len() < 3 {  
        return Err("Gerekli parametreler verilmedi...!");  
    }  
  
    let aranacak_deger = gelen_parametreler[1].clone();  
    let dosya_yolu = gelen_parametreler[2].clone();  
  
    let b_k_harf_duyarlı = env::var("IGNORE_CASE").is_ok();  
  
    Ok(Ayar {  
        aranacak_deger,  
        dosya_yolu,  
        b_k_harf_duyarlı,  
    })  
}
```

is_ok() → Çevre/Ortam değişkeni ayarlı değilse false döndürür. Bu fonksiyon için değer ne olduğu önemli değildir. Önemli olan ayarlanmış olmasıdır. Değer almak için unwrap(), expect() gibi result tipi döndüren bir yöntem kullanıyorduk.



Programı deneyince

```
cargo run -- file metin_dosyası.txt
```

boş değer verir çevre değişkeni ayarlı olmadığından

```
cargo run -- file metin_dosyası.txt
```

veri bulur.

ÇEVRE DEĞİŞKENİ İLE KULLANIMI

```
IGNORE_CASE=1 cargo run -- file metin_dosyası.txt
```

KÜÇÜK harf KULLANIMI İLE AYNI SONUÇLAR GELİR.

Power shell de kullanım

```
PS> $Env:IGNORE_CASE=1; cargo run -- to poem.txt
```

```
PS> Remove-Item Env:IGNORE_CASE → satırı ile unset edilir.
```



ÇIKTILARIN STANDART ÇIKTI YERİNE STANDART HATA ÇIKTISI İLE OLUŞTURMAK

Şimdiye kadar println!() makrosu ile çıktıları yazdık.

Çoğu terminal de iki tür çıktı vardır

Standard output (stdout – println)

→ genel bilgilendirme için

Standard error (stderr – eprintln)

→ hata mesajları için

Programın çıktısını başka bir dosyaya yazdırmak istediğimizde; **cargo run > output.txt**

eprintln!() → Hata mesajları ekrana yazılır, dosyaya yazılmaz.

println!() → Bütün mesajlar dosyaya yazılır.



```
fn main() {  
  
    let gelen_veriler: Vec<String> = env::args().collect();  
    let ayar = Ayar::insa_et(&gelen_veriler).unwrap_or_else(|hata| {  
        eprintln!("Argumanları alırken hata oluştu..! Hata Mesajı : --- {hata}--");  
  
        process::exit(1);  
    });  
  
    if let Err(e) = calistir(ayar) {  
        eprintln!("Uygulama hatası oluştu : -- {e} --");  
        process::exit(1);  
    }  
}
```

Şimdi hata olunca, hatalar dosyaya yazılmaz ekranda gözükür.

IGNORE_CASE=1 cargo run -- file metin_dosyası.txt > output.txt
hata olmadığında programın çıktısı dosyaya yazılacak ve ekrana yazılmayacak.



ITERATOR KULLANIMI : (Gelen verilere iteratör ile ulaşılması)

```
pub fn insa_et(
    mut gelen_parametreler: impl Iterator<Item = String>,
) -> Result<Ayar, &'static str> {
    gelen_parametreler.next();
    let aranacak_deger = match gelen_parametreler.next() {
        Some(deger) => deger,
        None => return Err("Gerekli parametreler verilmedi...!"),
    };
    let dosya_yolu = match gelen_parametreler.next() {
        Some(deger) => deger,
        None => return Err("Dosya yolu alınamadı ....!"),
    };
    let b_k_harf_duyarli = env::var("IGNORE_CASE").is_ok();
    Ok(Ayar {
        aranacak_deger,
        dosya_yolu,
        b_k_harf_duyarli,
    })
}
```



```
pub fn metin_ara<'a>(aranan_deger: &str, dosya_icerigi: &'a str) -> Vec<&'a str> {  
    dosya_icerigi  
        .lines()  
        .filter(|satir| satir.contains(aranan_deger))  
        .collect()  
}
```

```
pub fn metin_ara_case_insensitive<'a>(aranan_deger: &str, dosya_icerigi: &'a str) ->  
Vec<&'a str> {  
    let aranan_deger = aranan_deger.to_lowercase();  
    dosya_icerigi  
        .lines()  
        .filter(|satir| satir.to_lowercase().contains(aranan_deger))  
        .collect()  
}
```



ITERATOR KULLANIMI : (Komut satırı verilerinin iteratör - args()- ile alınması)

```
use command_line_arama::{calistir, Ayar};
use std::env;
use std::process;
fn main() {

    let ayar = Ayar::insa_et(env::args()).unwrap_or_else(|hata| {
        eprintln!("Argumanları alırken hata oluştu..! Hata Mesajı : --- {hata}--");

        process::exit(1);
    });

    if let Err(e) = calistir(ayar) {

        eprintln!("Uygulama hatası oluştu : -- {e} --");
        process::exit(1);
    }
}
```



<https://doc.rust-lang.org/book/ch12-00-an-io-project.html>



Celal AKSU

Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

<https://www.linkedin.com/in/cllaksu/>

<https://twitter.com/ksacil>

<https://www.youtube.com/@eemcs>