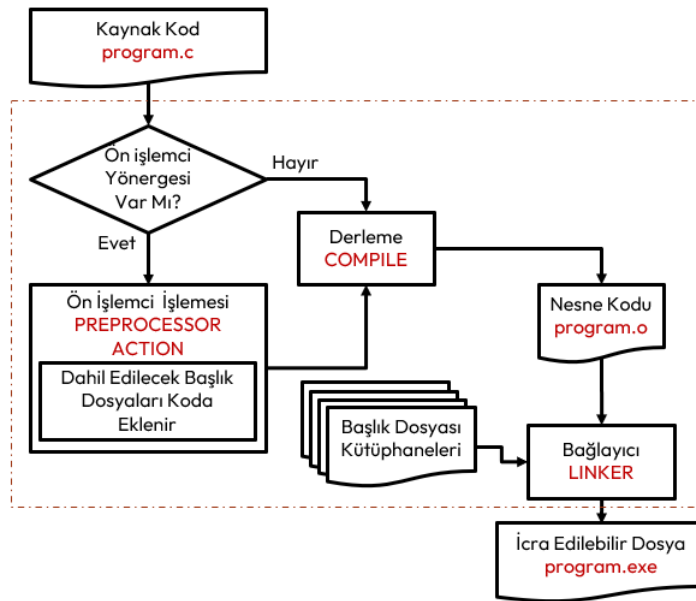


# ÖN İŞLEMCI YÖNERGELERİ

## Ön İşlemci Yönergesi Nasıl Çalışır

C dilinde **ön işlemci** (**preprocessor**) yönergeleri, derleyicinin bir parçası değildir, ancak derleme sürecinde ayrı bir adımdır. Ön işlemci, yalnızca bir metin değiştirme aracıdır ve gerçek derlemeden önce gerekli ön işlemeyi yapmasını sağlar. Kısaca derleme önce bul-değiştir mantığı ile kodda değişiklikler yaptırır.

- Ön işleme bir C kodunun derlenmesi öncesindeki ilk adımdır.
- Kodu **belirteçlere ayırma** (**tokenization**) adımından önce gerçekleşir.
- Ön işlemcinin önemli işlevlerinden biri de programda kullanılan kütüphane işlevlerini içeren başlık dosyalarını koda dahil etmek için kullanılmasıdır.
- Ön işlemci ayrıca değişmezleri (**literal**) tanımlar ve makro kullanımını sağlar.



Şekil 23. Derleme Süreci

C dilindeki ön işlemci ifadelerine, **yönergeler** (**directive**) denir. Programda ön işlemci bölümü her zaman C kodunun en üstünde görünür. Her ön işlemci ifadesi, **kare** (**hash**) **#** sembolüyle başlar. Aşağıda çok kullanılan yönergeler bulunmaktadır.

Yönerge	Açıklama
<b>#define</b>	Ön işlemci makrosunu değiştirmek ya da ilk defa tanımlamak için kullanılır.
<b>#include</b>	Bir başlık dosyasını diğer ile dahil etmek için kullanılır.
<b>#undef</b>	Tanımlanmış bir makroyu tanımsız hale getirmek için kullanılır.
<b>#ifdef</b>	Bir makro tanımlı ise doğru/true değerini döndürür.
<b>#ifndef</b>	Bir makro tanımlı değilse doğru/true değerini döndürür.
<b>#if</b>	Derleme zamanında bir durumun kontrolü için kullanılır.
<b>#else</b>	<b>#if</b> Yönergesinde alternatif durumu ifade eder.
<b>#elif</b>	<b>#else</b> ve <b>#if</b> yönergelerini tek bir şekilde ifade etmek için kullanılır.
<b>#endif</b>	Şart ön işlemcilerini ( <b>#if</b> , <b>#else</b> , <b>#elif</b> ) bitirir.
<b>#error</b>	<b>stderr</b> standart dosyasına hata mesajını yazar.
<b>#pragma</b>	Derleyiciye özel komutlar vermek için kullanılır.

Tablo 31. Ön İşlemci Yönergeleri

## Kullanıcı Tanımlı Başlık Dosyası

**#include** yönergesi ile; hazır olan başlık dosyalarını `< >` karakterleri arasında yazarak koda dahil ederiz. Hazır olmayan ve programcı tarafından hazırlanan başlık dosyalarını çift tırnak `" "` karakterleri arasında yazarak koda dahil ederiz.

**#define** yönergesi ile yeni bir makro tanımlanabileceği gibi tanımlanmış bir makro **#undef** ile ortadan kaldırabilir veya yenisini tanımlayabiliriz.

Aşağıdaki örnekte **PI** adlı bir makro tanımlanmış ve **3.1415** reel sayısını vermektedir. Daha önce **PI** adlı bir makro tanımlanmış ise derleyici hata verir.

```
#include <stdio.h>
#include "baslik.h"

#define PI 3.1415

#ifndef ENFAZLAOGRECISAYISI
    #define ENFAZLAOGRECISAYISI 100
#endif

#ifdef ENFAZLAOGRECISAYISI
    #undef ENFAZLAOGRECISAYISI
    #define ENFAZLAOGRECISAYISI 20
#endif
```

**DEBUG** hazır tanımlanmış bir makrodur ve hata ayıklama modunda kod derlenmesi halinde doğru/true değerini döndürür. Bunun için derleyiciye **-DDEBUG** argümanı verilir. **DEBUG** hazır tanımlanmış bir makro olup hata ayıklama modunda doğru/true olduğundan hata ayıklama modunda daha çok durum ve değer konsola yazılır. Bu durumda hatayı bulmak daha da kolaylaşır.

```
#ifdef DEBUG
/*
    Hata ayıklama modunda
    derleme yapıldığında
    çalışacak kod buraya yazılır. */
#endif
```

Aşağıda kullanıcı tanımlı bir başlık dosyası örneği verilmiştir; Başlık dosyasının bir koda birden fazla dahil (**include**) edilmesini önlemek için **\_BASLIK\_H\_** değişmezi, **şartlı (conditional)** olarak tanımlanmıştır. Bu başlık dosyası bir kod projesinde birden fazla dahil olması halinde, **\_BASLIK\_H\_** tanımlanmaz ise, başlık içindeki değişken ve fonksiyonlar birden fazla aynı kimlikle tanımlanacağından derleme yapılamayacaktır.

```
#ifndef _BASLIK_H_
#define _BASLIK_H_

#define PI 3.1415
#define ENFAZLAOGRECISAYISI 100

float* ogrenciNotlari() {
    static float notlar[ENFAZLAOGRECISAYISI];
    return notlar;
}

float ogrenciNotlarOrtalamasi(float* pNotlar, int pOgranciSayisi) {
    int sayac;
    float ortalama=0;
    for (sayac=0; sayac<pOgranciSayisi; sayac++) {
        ortalama+=pNotlar[sayac];
        #ifdef DEBUG
            printf("%d. Ogrencide Ortalama:%f\n", sayac, ortalama);
        #endif
    }
}
```

```
// Buradaki kod hata ayıklama modunda derlendiğinde çalışır.
// Bunun için derleyiciye -DDEBUG argümanı verilir
#endif
}
return ortalama/sayac;
}
#endif
```

Kendi hazırlamış olduğumuz başlık dosyasını (**baslik.h**) kodumuza dahil ettiğimizde artık başlık içindeki fonksiyon ve değişkenleri kullanabilir hale geliriz. Aynı klasörde/dizinde bulunacak şekilde bu başlık dosyasını kullanan örnek program aşağıda verilmiştir.

```
#include <stdio.h>
#include "baslik.h"

int main() {
    printf("PI=%f\n",PI);
    printf("En fazla öğrenci Sayısı=%d\n",ENFAZLAOGRECISAYISI);
    float* notlar=ogrenciNotlari();
    notlar[0]=100;
    notlar[1]=80;
    notlar[2]=60;
    float ortalama=ogrenciNotlarOrtalamasi(notlar,3);
    printf("%d öğrenci için Ortalama %f",3,ortalama);
}
```

## Ön Tanımlı Ön İşlemci Makroları

DEBUG gibi her c derleyicisi için standart olarak tanımlı makrolar bulunmaktadır;

Makro	Açıklama
__DATE__	Mevcut tarihi "MMM DD YYYY" biçiminde <b>dizgi (string)</b> olarak tanımlıdır.
__TIME__	Mevcut saat "HH:MM:SS" biçiminde <b>dizgi (string)</b> olarak tanımlıdır.
__FILE__	Dosya adı biçiminde <b>dizgi (string)</b> olarak tanımlıdır.
__LINE__	Dosyadaki satır numarası <b>tamsayı (int)</b> olarak tanımlıdır.
__STDC__	ANSI standardında derleme yapılıyorsa 1 olarak tanımlıdır.

Tablo 32. Standart Tanımlı Ön İşlemci Makroları

Bu makroları kullanan örnek program aşağıda verilmiştir;

```
#include <stdio.h>

int main() {
    printf("File: %s\n", __FILE__ );
    printf("Date: %s\n", __DATE__ );
    printf("Time: %s\n", __TIME__ );
    printf("Line: %d\n", __LINE__ );
    printf("ANSI: %d\n", __STDC__ );
}
```

## Parametrelili Ön İşlemci Makroları

Ön işlemci yönergeleriyle (**preprocessor directive**) tanımlanan makrolar daha derleme yapılmadan işlem görür. Dolayısıyla bu aşamada bazen parametrelerle işlem yapılması gerekebilir.

```
#define kare(x) ((x) * (x))
#define kup(x) ((x) * (x) * (x))
#define buyugu(x,y) ((x) > (y) ? (x) : (y))
```

Burada kullanılacak parametreler veri tipi tanımlanmaz. Dolayısıyla kullanılacağı yerlere buna dikkat edilerek parametrelili makro tanımlanmalıdır. Eğer makro birkaç satırdan oluşacak ise **ters bölü (back slash) \** karakteri kullanılır.

```
#include <stdio.h>

#define MAKRO(num, str) { \
    printf("%d", num); \
    printf(" is"); \
    printf(" %s number", str); \
    printf("\n"); \
}
```

## Derleme Sürecinin Detayı

C dilinde **derleme** (**compile**) sürecinin iki aşamadan oluştuğu daha önce anlatılmıştı. Aşağıda tüm süreç verilmiştir;

1. Birinci aşama:
  - a. Kaynak koddaki tüm açıklamalar silinir.
  - b. **#define**, **#if**, **#include**, gibi tüm yönergeler ile verilen işlemler yapılır. Bunlar arasında başlık dosyalarının koda dahil edilmesi de vardır.
  - c. Bu aşamaya gelen kod, **gcc -E main.c** komutu yazılarak görülebilir.
2. İkinci aşama:
  - a. Kaynak kod **montaj koda** (**assembly code**) çevrilir.
  - b. Bu duruma gelen kod, **gcc -S main.c** komutu dosya haline getirilebilir.
  - c. Assembly kod derlenerek makine diline çevrilir ve **amaç dosya** (**object file**) oluşur.
  - d. Bu duruma gelen kod, **gcc -c main.c** komutu ile dosya haline getirilebilir.
  - e. Amaç dosya kodda belirtilen başlık dosyalarına uygun **kütüphaneler** (**library**) ile birbirine bağlanır ve **icra edilebilir dosya** (**executable file**) elde edilir. Burada bahsedilen kütüphaneler her işletim sistemi için ayrı olarak oluşturulur ve derleyici kurulumunda bir klasörde tutulur.
  - f. Şimdiye kadar yapılan tüm derlemelerde yukarıdaki adımlar atlanarak **gcc main.c -o main.exe** komutuyla icra edilebilir dosya oluşturulmaktadır.