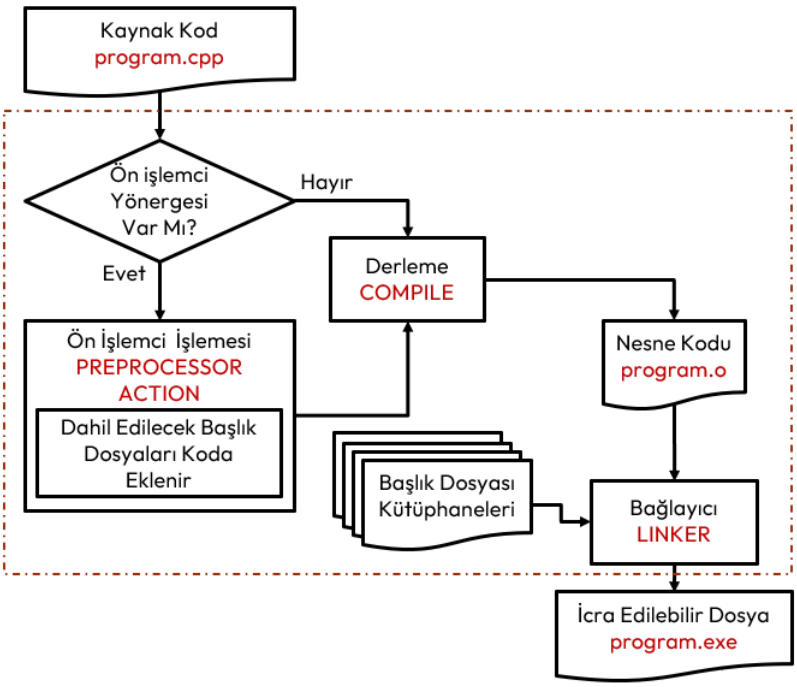
ÖN İŞLEMCİ YÖNERGELERİ

Ön İşlemci Yönergesi Nasıl Çalışır

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

- Ön işleme bir C++ kodunun derlenmesindeki ilk adımdır.
- Kodu sembollere 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 sabitleri tanımlar ve makro kullanımını sağlar.



Şekil 57. 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, karma (hash) sembolüyle başlar. Aşağıda en çok kullanılan yönergeler bulunmaktadır.

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

Tablo 26. 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 <iostream>
#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; Baslık dosyasının bir koda birden fazla dahil (include) edilmesini önlemek için `_BASLIK_H_` sabiti 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 yapılamayacaktı.

```
#ifndef _BASLIK_H_
#define _BASLIK_H_

#include <iostream>
using namespace std;

#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
        cout << sayac << ". Ogrencide Ortalama: " << ortalama endl;
        // Buradaki kod hata ayıklama modunda derlendiğinde çalışır.
        // Bunun için derleyiciye -DDEBUG argümanı verilir
    #endif
  }
  return ortalama/sayac;
}
#endif
```

Bütün bu `#ifndef`, `#define` ve `#endif` yönergeleri ile yapılmak isteneni `#pragma once` yönergesi ile aşağıdaki gibi yapabiliriz.

```
#pragma once

#include <iostream>
using namespace std;

#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
        cout << sayac << ". Ogrencide Ortalama: " << ortalama endl;
        // Buradaki kod hata ayıklama modunda derlendiğinde çalışır.
        // Bunun için derleyiciye -DDEBUG argümanı verilir
    #endif
  }
  return ortalama/sayac;
}
```

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 <iostream>
#include "baslik.h"

using namespace std;

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);
    cout << "3. öğrenci için Ortalama: " << ortalama << endl;
}
```

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 dizgi (string) olarak tanımlıdır.
`__TIME__`	Mevcut saat "HH:MM:SS" biçiminde dizgi (string) olarak tanımlıdır.
`__FILE__`	Dosya adı biçiminde dizgi (string) olarak tanımlıdır.
`__LINE__`	Dosyadaki satır numarası tamsayı (int) olarak tanımlıdır.
`__STDC__`	ANSI standardında derleme yapılıyorsa 1 olarak tanımlıdır.

Tablo 27. Standart Tanımlı Ön İşlemci Makroları

Bu makroları kullanan örnek program aşağıda verilmiştir;

```
#include <iostream>
using namespace std;

int main() {
    cout << "File: " << __FILE__ << endl;
    cout << "Date: " << __DATE__ << endl;
    cout << "Time: " << __TIME__ << endl;
    cout << "Line: " << __LINE__ << endl;
    cout << "ANSI: " << __STDC__ << endl;
}
```

Parametreli 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 parametreli makro tanımlanmalıdır. Eğer makro birkaç satırdan oluşacak ise ters bölü (back slash) `\` karakteri kullanılır.

```
#include <iostream>
using namespace std;

#define MAKRO(num, str) { \
            Cout << num  \
                 <<" is") \
                 << str << " number" \
                 << endl; \
           }
```

Derleme Sürecinin Detayı 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, `g++ -E main.cpp` komutu yazılarak görülebilir.
2. İkinci aşama:
   a. Kaynak kod montaj koda (assembly code) çevrilir.
   b. Bu duruma gelen kod, `g++ -S main.cpp` komutu dosya haline getirilebilir.
   c. Assembly kod derlenerek makine diline çevrilir ve amaç dosya (object file) oluşur.
   d. Bu duruma gelen kod, `g++ -c main.cpp` 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 `g++ main.cpp -o main.exe` komutuyla icra edilebilir dosya oluşturulmaktadır.