**C ile Programlamada Ön işlemci Komutları(Preprocessor Commands)**

Önişlemci komutları C dilinde oldukça önemli olan Komutlardandır. Bunlarla ilgili bilinmesi gereken en önemli şey bu komutların derleme zamanı sırasında yapılmasıdır. Şuana kadar ezbere kullandığımız kütüphane eklemek için kullanılan include önişlemci komutu sadece standart kütüphaneden başlık dosyalarını değil aynı zamanda bizim yazmış olduğumuz başlık dosyalarını da kaynak dosyaya ekleyebiliriz. C dilindeki önişlemci komutları şu şekilde sıralanabilir:

1-)#include önişlemci komutu

Bu ön işlemci komutu kod derlenirken include ile bildirilmiş olan header files(Başlık Dosyaları) derleme zamanında kaynak dosyaya yapıştırılır. Bu sayede include ile eklenmiş olan kütüphanelerdeki fonksiyonlar doğrudan kullanılabilir. Daha önce de belirtildiği gibi bu eklenen başlık dosyalarının standart kütüphaneye ait olması zorunluluğu yoktur. Örneğin aşağıdaki örnekte NumberUtil programcının kendi yazdığı bir header filedır:

#include “NumberUtil.h”

int main(void)

{

//NumberUtil.h içerisindeki fonksiyonlar kullanılabilir.

}

Şüphesiz bu header file standart kütüphaneden de olabilir. Aynı aşağıdaki örnekte gösterildiği gibi :

#include <stdio.h>

int main(void)

{

int val;

printf(“Bir sayi giriniz?\n”);//stdio.h’nın bir fonksiyonu

scanf(“%d”,&val);// stdio.h’nın bir fonksiyonu

printf(“Girdiginiz sayi : %d”,val);

}

Daha önceden sıkça kullandığımız printf ve scanf fonksiyonları include ön işlemci komutu kullanılarak stdio.h başlık dosyasının eklenmesiyle kullanabilir hale getirilmektedir. Programcının kendi oluşturduğu modüllerdeki fonksiyonlar aynı bu şekilde bu dosya içeriğinin bu ön işlemci komutuyla birlikte kullanılmasıyla kaynak dosyaya derleme zamanında yapıştırılır. Kısaca kullanmak için gereken proses bu şekilde gerçekleşir.

2-)#define ön işlemci komutu

Bu komut derleme zamanında sembolik adlar(makrolar) tanımlamak için kullanılır. Tanımlanan sembolik adlar programın herhangi bir yerinde kullanılabilir. Makrolar fonksiyonel makro da olabilir. Bir sabit makro da olabilir. Bir sabit makronun tanımlanması aşağıdaki gibi yapılabilir:

#define SIZE 100

int main(void)

{

int arr[SIZE];//geçerli

}

Kodun geçerli olmasından da anlaşılabileceği gibi burdaki size değeri bir sabittir. Derleme zamanı içerisinde kaynak kod içerisinde SIZE görülen her yere 100 yaz anlamına gelmektedir. Bu sebeple dizi boyutu için doğrudan kullanımı geçerlidir.

Fonksiyonel makrolar da normal fonksiyonların bir alternatifi olarak kullanılabilir. Özellikle bazı basit işlemlerin yapılması için oldukça efektiftir. Örneğin:

#include <stdio.h>

#define Max(a,b) (a > b ? a : b)

int main(void)

{

int a,b;

printf("iki sayi giriniz?\n");

scanf("%d%d",&a,&b);

printf("Max(%d,%d) : %d",a,b,Max(a,b));

}

3-) #if,#ifdef,ifndef,#else ve #endif ön işlemci komutları

Bu ön işlemci komutlarından üretimde en sık kullanılanlar header file içerisinde ifndef ön işlemci komutunun daha önceden anlattığımız define komutu ile birlikte kullanımıdır. Bu syntax header file’ın tek bir defa include ön işlemci komutuyla kaynak dosyaya eklenmesini sağlamaktadır. Bu duruma multiable inclusion guard denilmektedir. Üretimininde bir Header file genellikle şu şekildedir:  
#ifndef NUMBERUTIL\_H

#define NUMBERUTIL\_H

//Fonksiyon prototipleri

#endif

Bu header fileların hepsinin mutlaka bir .c uzantılı kaynak dosyası bulunur. Header file içerisinde bildirimleri olan bu fonksiyonların .c uzantılı kaynak dosyasında fonksiyonların gövdeleri bulunur. Bu bir C programının parçaları ayrılması için kullanılan en önemli sentakstır. Diğer taraftan ifndef ön işlemci komutunun farklı kullanım alanları da olabilir. Örneğin bazı kodlar sistem bağımsız hale getirilmek istenebilir. Mesela Linux işletim sisteminde şu şekilde çalış Windows veya başka işletim sisteminde şu şekilde çalış ya da release modunda şu şekilde çalış debug modunda şu şekilde çalış ve benzeri kodlar yazılabilir. Aşağıda bunlarla ilgili birkaç örnek verilmiştir:

#ifdef DEBUG

//Debug modu için şu kodları yürüt

#else

//Debug dışındaki modlar için bu kodları yürüt

#endif

Anahtar Notlar: if ile başlayan önişlemci komutlarında mutlaka bir endif kısmı bulunmalıdır. Aksi durumda error oluşur. Örneğin

#if

//kod

#else

//kod

//error endif yok

else ön işlemci komutu if deyiminin else kısmı gibi çalışır. Ancak aradaki en önemli fark bunun her zaman derleme zamanı içerisinde yapılmasıdır. Bu sebeple hem optimizasyonu hem de okunabilirlik ve algılanabilirliği doğrudan etkiler. Aşağıdaki örnekte ise bir ifade define edilmiş mi kontrolü STM32 kartına göre belirlenmektedir:

#ifdef STM32

//stm32 kartına göre olan kodu derle

#else

//diğer kartlara göre olan kodu derle

#endif

Bu verilen örnekler dışında da elbette başka kullanım alanları da bulunmaktadır. Ancak üretimde en çok kullanılan konseptler burda verilen örneklerdir. Şüphesiz ön işlemci programın parçalara bölünmesinde oldukça önemli bir yer sahiptir. Çoklu dosyalarla çalışırken temelde genel biçim olarak aşağıdaki bir kod yazılabilir:

//main fonksiyonun bulunduğu dosya(main.c)

#include “nutility.h”

int main(void)

{

//nutility içerisindeki fonksiyonlar burda kullanılabilir.

}

//nutility.h dosyası veya modülü

#ifndef NUTILITY\_H

#define NUTILITY\_H

//fonksiyon prototipleri

#endif

//nutility.c dosyası veya modülü

//fonksiyonları gövde kısımları

//bu modülde bulunmalıdır.

Yukarıdaki örnekte aslında bir c programı nasıl modüllere ayrılır ve okunabilirlik ve algılanabilirlik açısından iyi hale getirilir. Bunun için öncelikle fonksiyonlarla parçalara bölmüştük ama bu ön işlemci komutlarıyla birlikte aslında profesyonel düzeyde bir programın nasıl oluşturması gerektiği anlamış olduk. Tek bir fonksiyonla çalışılmayacağı gibi tek bir dosyayla veya modülle de çalışılmayacağını aslında görmüş.