NOT 11
Ön İşlemci Komutları

Bu komutlar program geliştirmede önemli etkiye sahiptir.*11.15.Derleme Seçenekleri* başlığında örnek bir kullanım oluşturulmuştur.

- C derleyicileri iki ayrı birimden oluşur:
  - Ön işlemci Birimi
  - Derleme Birimi
- Ön işlemcinin, bilgisayarın işlemcisi ya da başka bir donanımsal elemanıyla hiçbir ilgisi yoktur.
- Ön işlemci, belirli bir iş gören bir yazılım programıdır.
- Ön işlemci, kaynak dosya üzerinde birtakım düzenlemeler ve değişiklikler yapan bir ön programdır.
- C programlama dilinde # ile başlayan bütün satırlar, ön işlemci programa verilen komutlardır (directives).

#include	*#elif*	*#undef*
#define	*#ifdef*	*#line*
#if	*#ifndef*	*#error*
#else	*#endif*	*#pragma*

11.1.include

`#include` komutu ile, ismi verilen dosyanın içeriği, bu komutun yazıldığı yerden itibaren sanki içeriği yapıştırılmış gibi tüm tanımlarıyla birlikte kullanıma açılır.
`#include` ön işlemci komutu kaynak programın herhangi bir yerinde bulunabilir. Fakat standart başlık dosyaları gibi, içinde çeşitli bildirimlerin bulunduğu dosyalar programın ilk satırlarına yazılır.
`#include` komutu, iç içe geçmiş bir biçimde de bulunabilir. Örneğin çok sayıda dosyayı kaynak koda direk eklemek yerine başka bir kitaplık dosyası yazılarak sadece o dosya eklenerek kullanılır.

Bu işlemcinin genel olarak iki şekilde kullanılır.

1. `#include <kitaplık.h>`
   Bu tip kullanım derleyicinin standart kitaplık klasöründe bulunan veya bu klasöre sonradan eklenmiş kitaplık dosyalarının kullanılması için kullanılır.
   Klasör adı (windows'ta \ Unix'te/) ayıraç dosya adı şeklinde kullanılabilir.

`#include <stdio.h>` `#include <GL/gl.h>`	`#include <stdlib.h>` `#include <GL\gl.h>`

2. `#include "kitaplık.h"`
   Çift tırnak operatörü kullanımı herhangi bir klasör altında bulunan kitaplık dosyalarının eklenmesi için kullanılır.
   Dosya yolu ayıraç klasör adı ayıraç dosya adı şeklinde kullanılabilir.

`#include "deneme.h"`	`#include "C:\Users\machine\Documents\Proje\file\deneme.h"`

11.2.define

`#define` komutu ile genel sabit değerler (simgesel değişmez) veya işlevler (basit makro) tanımlanır.
Bu satır işlenirken derleyici tanımı değeriyle yer değiştirdikten sonra derleme işlemini gerçekleştirir.
Bu ifade kodda genişlemeye sebep olur.
Bu işlemci kullanım olarak `#define boşluk tanım boşluk değer` şeklinde kullanılır.
Makro tanımı ise `#define boşluk tanım(değişken) boşluk ((değişken)işlem)`

Yazım	Eş değeri

`#define boy 8`
`#define en (boy-3)`
`#define and(x,y) ((x)&&(y))`
`printf("%d",and(en,boy));`
	`printf("%d",((8-3) && (8));`

Bu komut sayesinde kod okunurluğu arttırılır. Ayrıca sabit bir değişken birden fazla yerde kullanıyor ise bu değişkenin değiştirilmesi için tüm kullanılan yerlerden silinmesi gerekir. Bunun yerine `#define` ile tanımlanırsa sadece tanımın değerinin değişmesi bu işlemi kolay bir şekilde yapar.

11.3.if

Koşullu derleme komutudur. Yazılan kaynak kodun belirli kısımları ön işlemci program tarafından derleyiciye verilirken bazı kısımları derleyiciye verilmez. Derleyiciye verilecek kaynak kod ön işlemci program tarafından seçilir.
if koşuluyla aynıdır.
Endif ile sonlanır.
Var değişkeni 8 den büyük olursa programı sonlandır.

```
#define var 10
#if var>8
printf("program sonu");
return 0;
#endif // var
```

```
#define var 1
#if var>8
printf("program sonu");
return 0;
#endif // var
```

11.4.else

Koşullu derleme komutudur.
İf ten sonra kullanılır.
İşlevi else ile aynıdır.
Tanımlanan sayı 5 ten büyükse büyük, küçükse küçük yazan satırlar.

```
#define var 6
#if var>5
printf("Buyuk");
#else
printf("Kucuk");
#endif // var
```

```
#define var 5
#if var>5
printf("Buyuk");
#else
printf("Kucuk");
#endif // var
```

11.5.elif

Koşullu derleme komutudur.
İf ten sonra kullanılır.
işlevi else if ile aynıdır.
Tanımlanan sayı 5 ten büyükse büyük, küçükse küçük ve eşitse eşit yazan satırlar.

```
#define var 6
#if var>5
printf("Buyuk");
#elif var==5
printf("esit");
#else
printf("Kucuk");
#endif // var
```

```
#define var 5
#if var>5
printf("Buyuk");
#elif var==5
printf("esit");
#else
printf("Kucuk");
#endif // var
```

```
#define var 4
#if var>5
printf("Buyuk");
#elif var==5
printf("esit");
#else
printf("Kucuk");
#endif // var
```

11.6.ifdef

Koşullu derleme komutudur.

`#if defined` ile aynı anlamlıdır.

Bir tanımın `#define` ile yapılıp yapılmadığını kontrol eder.

Tanımlanmışsa işletilir.

```
#define var 5
#ifdef var
printf("%d",var);
#endif // var
```

```
5
```

11.7.ifndef

Koşullu derleme komutudur.

`#if !defined` ile aynı anlamlıdır.

Bir tanımın `#define` ile yapılıp yapılmadığını kontrol eder.

Tanımlanmamışsa işletilir.

Başlık dosyalarında birden fazla eklenmesin diye sıklıkla kullanılır.

```
#ifndef DENEME_H_INCLUDED
#define DENEME_H_INCLUDED
/**fonksiyon prototipleri
ve diğer tanımlar*/
#endif
```

11.8.endif

Tüm if ifadeleri endif le sonlandırılır.

11.9.undef

Define ile önceden tanımlanmış bir ifadenin iptal edilmesi için kullanılır.

```
#define a 5
……
#undef a
//bu satırdan itibaren tanımsız
```

```
#define a 5
……
#undef a
//bu satırdan itibaren tanımsız
#define a 6
//bu satırdan itibaren değeri 6
```

11.10.line

Bu ön işlemci komutuyla derleyicinin kaynak koda ilişkin tuttuğu satır numarası ve dosya ismi değiştirilebilir. Bu komut iki ayrı değer alabilir. Komutun alabileceği birinci değer bir tamsayı olarak satır numarasıdır. Komutun isteğe bağlı olarak alabileceği ikinci değer dosya ismidir. Bu ön işlemci komutu kaynak kod üreten programlar tarafından kullanılabilir.

__LINE__ ve *__FILE__* tanımlarına etki eder.

```
printf("Dosya %s %d.Satir\n", __FILE__,
__LINE__);
#line 4
printf("Dosya %s %d.Satir\n", __FILE__,
__LINE__);
#line 5 "file.c"
printf("Dosya %s %d.Satir\n", __FILE__,
__LINE__);
```

```
Dosya C:\islem_line\main.c 12.Satir
Dosya C:\islem_line\main.c 4.Satir
Dosya file.c 5.Satir
```

11.11.error

`#error` ön işlemci komutu koşullu derlemede kullanılır.

Ön işlemci program bu komut ile karşılaştığında derleme işlemini daha önişleme aşamasında sonlandırır. Kullanımı aşağıdaki gibidir:

```
#error Derlenme Modu Tanımlanmamış
```

11.12.pragma

Derleyiciye komut vermeye yarar ve derleyiciden derleyiciye değişiklik gösterir.
örnek olarak GCC de aşağıdaki komut derleme ekranında
main.c|12|note: #pragma message: Derleniyor main.c...| gibi bir çıktı üretir.

```
#pragma message "Derleniyor " __FILE__ "..."
```

11.13.Ön Tanımlı İfadeler

Standart C dilinde 5 tane simgesel değişmez önceden tanımlanmış kabul edilir. Herhangi bir başlık dosyası içinde bu simgesel değişmezler #define ön işlemci komutuyla tanımlanmış olmamasına karşın kaynak kodun derleyici tarafından ele alınmasından önce bir yer değiştirme işlemine sokulurlar. Bu simgesel değişmezler çoğunlukla hata arama amacıyla yazılan kodlarda kullanılırlar:

__LINE__

Bu simgesel değişmez kaynak kodun kaçıncı satırında kullanılmış ise, o satırın numarasını gösteren bir tamsayı ile yer değiştirilir.

__FILE__

Bu simgesel değişmez hangi kaynak dosya içinde kullanılmış ise, o kaynak dosyanın ismini gösteren bir metinle yer değiştirilir.

__DATE__

Bu simgesel derleme tarihini gösteren bir metinle yer değiştirilir. Tarih bilgisini içeren yazının formatı yandaki gibidir: aa gg yyyy (ay, gün, yıl)

__TIME__

Bu simgesel değişmez derleme zamanını gösteren bir metinle yer değiştirilir. Zaman bilgisini içeren yazının formatı yandaki gibidir: sa:dd:ss (saat, dakika, saniye)

__STDC__

Eğer derleyici standart C derleyicisi ise bu simgesel değişmez tanımlı kabul edilir. Derleyici standart C derleyicisi değil ise bu simgesel değişmez tanımlanmamış kabul edilir.

Aşağıdaki programı derleyerek çalıştırın:

```
printf("derleme tarihi = %s\n", __DATE__);
printf("derleme zamani = %s\n", __TIME__);
printf("bu satirin numarasi = %d\n", __LINE__);
#ifdef __STDC__
printf("standart C derleyicisi\n");
#else
printf("standart C derleyicisi degil\n");
#endif //   STDC__
```

```
derleme tarihi = Oct 12 2015
derleme zamani = 16:29:58
bu satirin numarasi = 9
standart C derleyicisi
```

11.14.Makro

Makro, fonksiyon tanımlamadan fonksiyonun getirdiği bazı avantajlardan yararlanması için yapılan bir yazım şeklidir. Ortada gerçek bir fonksiyon olmadığından program çalışırken dallanma olmaz.
Makrolar açılım yapıldığından düzgün tanımlanmalıdır.
1)Makrolar çağrılırken matematiksel işlem yapılmaz. Bu yüzden işlevsel öncelik sorunları ortaya çıkar. Bu tür öncelik sorunlarını çözmek amacıyla makro açılım listesi dıştan öncelik ayracı içine alınmalıdır.

`#define kare(x) x*x`
`int x=3;`
`printf("%d karesi %d",x+2,kare(x+2));`
	`#define kare(x) (x)*(x)`
	`int x=3;`
	`printf("%d karesi %d",x+2,kare(x+2));`

Bu kullanımda sonuç 3+2*3+2=11	Bu kullanımda sonuç (3+2)*(3+2)=25

`#define kare(x) (x)*(x)`
`int x=4;`
`printf("%d",kare(x++));`
	Bu kullanımda (x++)*(x++) işlemi yapıldığından Sonuç 4*5=20 değerini alır.

2)Makrolar çağrılırken değişken olarak fonksiyonlar kullanılması fonksiyonun iki kez çağrılmasına neden olur. Makro yerine fonksiyon içinde fonksiyon kullansaydık fonksiyondan dönen değer diğer fonksiyonun parametresi olurdu.

```
#define kare(x) (x)*(x)
int x=4;
printf("%d",kare(fonksiyon()))
;
```

```
printf("%d",((fonksiyon())*(fonksiyon())))
;
```

3) Bir makro tanımının açılım listesinde yer alan # atomu ön işlemci programın bir işlecidir.
işleci önek konumunda tek terimli bir işleçtir. Bu işlemcinin terimi makro parametrelerinden biri olmalıdır. Parametresi olmayan makrolarda bu işleç kullanılamaz.# tanımı değişkeni iki çift tırnak arasına alır.

```
#define printint(x) printf(#x " = %d\n", x);

printint(0xa);
```

```
printf("0xa" " = %d\n", 0xa);
```

```
#define print(x) printf(#x);

print(tirnak ekler\n);
print("tirnak ekler");
```

```
printf("tirnak ekler\n");
printf("\"tirnak ekler\"");
```

4)## birleştirici anlamındadır. Girilen ifadeleri birleştirir ve isim yapar;

```
#define bir(x,y) x##y

int bir(a,b);
ab=8;
printf("ab=%d",ab);
```

```
int ab;
ab=8;
printf("ab=%d",ab);
```

5)Bir fonksiyonla aynı isimde olması

```
inline int kare(int a)
{
    return a*a;
}
#define kare(x) (x)*(x)
int main()
{
    printf("%d",kare(5));
    return 0;
}
```

Böyle bir kullanımda ön işlemci derlemeden önce çalıştığından kare fonksiyonu yerine kare makrosu çalıştırılmış olur.

```
printf("%d",(5)*(5));
```

Fonksiyonun çağrılması isteniyorsa `(kare)(5)` isim parantez içine alır ya da kullanılmadan önce `#undef kare` yazılması gerekir.

11.15.assert Makrosu

```
#include <assert.h>
```

Programdaki yanlış çalışmaya yol açan gizli hatalara böcek bunları bulmaya böcek ayıklama (Debug) denilmektedir. Yapılan araştırmalar, programcının yaptığı hataların 5% ile % 8 arasının son ürüne yansıdığını göstermektedir. Ayrıca böceklerin ürün satıldıktan sonra düzeltilmesi çok maliyetli ve üreticinin başarısını gölgelemektedir. Bu yüzden böcek oluşumunun en başta proje yöneticisi tarafından planlanıp, programcı tarafından engellenmesi sağlanmaktadır.

Programcı kodunu yazarken aynı zamanda da bir böcek oluşuyor mu diye kontrol etmelidir. Bu işlem sıklıkla assert makrosu ile yapılır. assert makrosu assert.h kitaplığında tanımlanmış olup assert(ifade) şeklinde kullanılır. Kullanılmadan önce `#define NDEBUG` tanımlanmamış olmaması gereklidir.

Programcı assert makrolarını kodun çeşitli yerlerine yerleştirerek hata durumlarının oluşmasını gözler. Fakat birden fazla işlemle(multi thread) çalışıldığında hata ayıklama için farklı yöntemler düşünülmelidir.

Aşağıdaki örnekte bir işaretçi null değerinde bırakılmıştır. Böyle bir işlem Windows ta program sonlandırıldı hatası verirken Unix tabanlı sistemlerde segmentation fault hatası verir.

```
char *a=NULL;
assert(a);
```

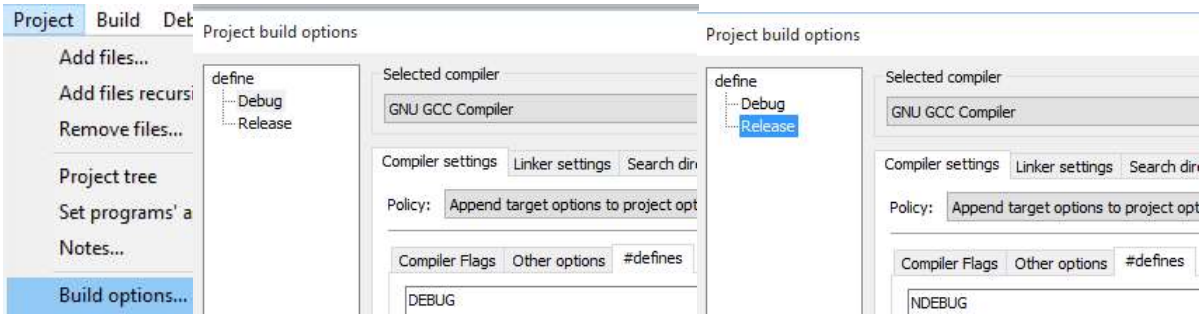
```
Assertion failed: a!=NULL, file C:\test\main.c,line 30

This application has requested the Runtime to terminate it in an unusual
way.
Please contact the application's support team for more information.

Process returned 255 (0xFF) execution time : 11.454 s
Press any key to continue.
```

11.16.Derleme Seçenekleri

Koşullu derleme komutları sıklıkla debug(Böcek ayıklamak) amacıyla kullanılır.
Program yazılırken hata ayıklama amacıyla programa bazı kodlar eklenir.
Programın son sürümünde hata aramaya yönelik kodların bulunması, programın çalışma zamanında getireceği ek maliyetten ve kullanıcının hatayı görmesinden dolayı istenmez.
Bu yüzden bir program yazılırken hata ayıklama modunda geliştirilir.
GCC derleyicinse derlerken –DDEBUG veya –DNDEBUG tanımları kullanılabilir. –D `#define` anlamındadır.
Code::Blocks ta ise bu işlem Project->Build options… Seçilerek build options penceresinin #defines sekmesi üzerinden gerçekleştirilir.



Aşağıdaki örnek hata ayıklama modunda parametreleri klavyeden alırken, son sürüm modunda konsoldan parametre alır. Aşağıda örneği bu tanımları yaparak derleyelim.

Derleme seçimi bu menüden yapılır. Ve seçilen işleme göre derlenen çıktı <Proje yolu>bin yolunda ilgili klasörde bulunur.

```c
#include <stdio.h>
#include <string.h>
#ifdef NDEBUG
///release sürümü
int main(int girdi,char*komut[]){
#elif DEBUG
///debug sürümü
int main(){
    char komut[100][200];
    strcpy(komut[0],"C:\\programfiles\\a.exe");
    int girdi=1;
    char c=0;
while(scanf("%s%c",komut[girdi++],&c)&&c!='\n');
#else
#error Derlenme Modu Tanımlanmamış
#endif // NDEBUG
    printf("Girdi sayisi %d\n",girdi);
    int i;
    for(i=0; i<girdi; i++)
        printf("%s\n",komut[i]);
    return 0;}
```

Hata Ayıklama

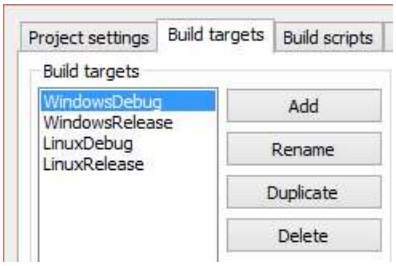
```
a b c
Girdi Sayisi 4
C:\programfiles\a.exe
a
b
c
```

Son Sürüm

```
\bin\Release>define.exe a
b c

Girdi Sayisi 4
define.exe
a
b
c
```

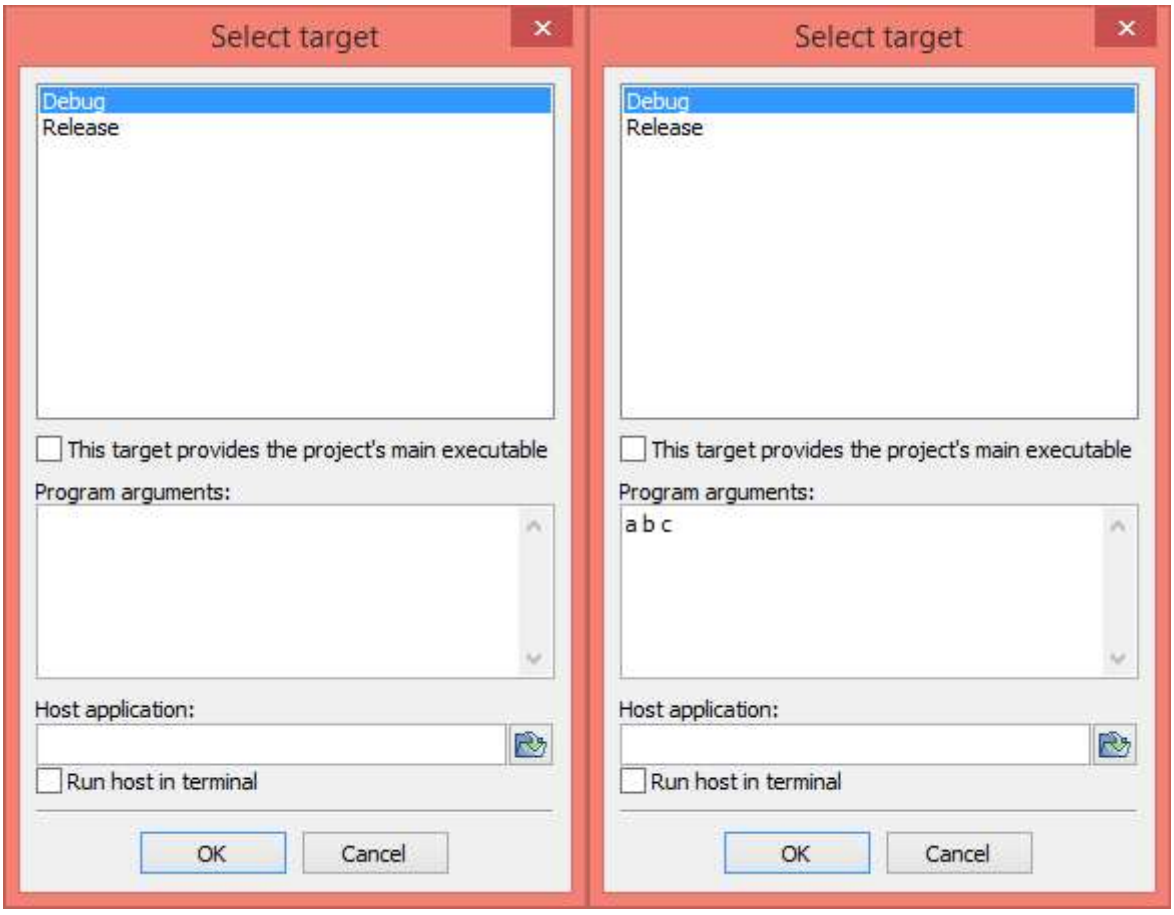
Derleme seçenekleri değiştirilebilinir ve arttırabilinir.Bu işlem için Project->Properties.. seçeneğinden açılan penceredeki Build targets sekmesinden yapılır.



11.17.Çalıştırma Seçenekleri

Program IDE üzerinden çalıştırılken girdi kabul eden main fonksiyonuna değişken değerleri gönderilebilir.Bu işlem için Project->Set programs' arguments... seçeneğinden açılan penceredeki hedef seçeneklerden istenen program Arguments kısımın yazılır.Aşağıdaki örneği çalıştıralım.

```
#include <stdio.h>
int main(int komut_sayisi,char *komutlar[])
{
 printf("Girdi sayisi=%d,komut sayisi=%d\n",komut_sayisi,komut_sayisi-1);
 printf("Verilen komutlar\n");
 int i=0;
 printf("Dosya yolu ve adi\t%s\n",komutlar[0]);
    for(i=1; i<komut_sayisi; i++)
        printf("%d.komut\t%s\n",i,komutlar[i]);
    return 0;
}
```



```
Girdi sayisi=4,komut sayisi=3
Verilen Komutlar
Dosya yolu ve adi    Debug\giris.exe
1.Komut a
2.komut b
3.komut c
```