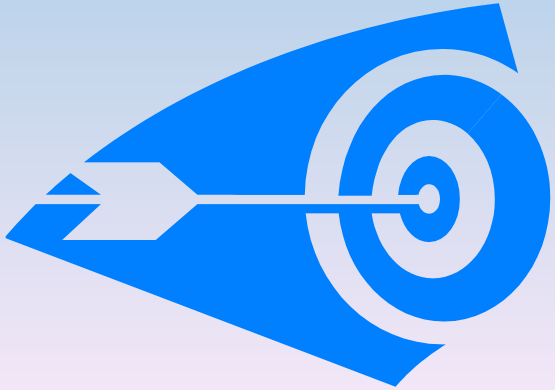
İstanbul Üniversitesi
Elektrik Elektronik Mühendisliği

Önişlemci Komutları, switch ve goto deyimleri



Kaynak: C ve Sistem Programcıları Derneği Kurs notu

Yrd.Doç.Dr.Mahmut YALÇIN

ÖNİŞLEMCİ KOMUTLARI (1)

C derleyicileri iki ayrı modülden oluşur:

1. Önişlemci Modülü
2. Derleme Modülü

Önişlemcinin, bilgisayarın işlemcisi ya da başka bir donanımsal elemanıyla hiçbir ilgisi yoktur. Önişlemci, belirli bir iş gören bir yazılım programıdır.

Önişlemci, kaynak dosya üzerinde birtakım düzenlemeler ve değişiklikler yapan bir ön programdır. Önişlemci programının bir girdisi bir de çıktısı vardır. Önişlemcinin girdisi kaynak dosyanın kendisidir. Önişlemci programın çıktısı ise derleme modülünün girdisini oluşturur. Yani kaynak program ilk aşamada önişlemci tarafından ele alınır. Önişlemci modülü, kaynak dosyada çeşitli metinsel düzenlemeler, değişiklikler yapar. Daha sonra değiştirilmiş ya da düzenlenmiş olan bu kaynak dosya, derleme modülü tarafından amaç koda dönüştürülür.

C programlama dilinde **#** *ile başlayan bütün satırlar, önişlemci programa verilen* komutlardır (*directives).* Önişlemci program, önceden belirlenmiş bir komut kümesindeki işlemleri yapabilir. Her bir komut, **#** atomunu izleyen bir sözcükle belirlenir.

Önişlemci program, amaç kod oluşturmaya yönelik hiçbir iş yapmaz, kaynak kod içinde bazı metinsel düzenlemeler yapar. Kendisine verilen komutları yerine getirdikten sonra, **#** ile başlayan satırları kaynak dosyadan siler. Derleme modülüne girecek programda **#** ile başlayan satırlar artık yer almaz.

#include Önişlemci Komutu

Bu önişlemci komutunun genel sözdizimi aşağıdaki gibidir:

#include <dosya ismi>

ya da

#include "dosya ismi"

#include komutu ile, ismi verilen dosyanın içeriği, bu komutun yazıldığı yere yapıştırılır. Bu komut ile önişlemci, belirtilen dosyayı diskten okuyarak komutun yazılı olduğu yere yerleştirir. Bu komutla yapılan iş, metin düzenleyici programlardaki "kopyala - yapıştır“ (*copy – paste) işlemine benzetilebilir.*

#include önişlemci komutuyla, kaynak dosyaya eklenmek istenen dosyanın ismi iki ayrı biçimde belirtilebilir:

1. Açısal ayraç içinde:

#include <stdio.h>

#include <time.h>

2. Çift tırnak içinde

#include "general.h"

#include "genetic.h"

Dosya ismi eğer açısal ayraç içinde verilmişse, sözkonusu dosya önişlemci tarafından, yalnızca önceden belirlenmiş bir dizin içinde aranır.

#define Önişlemci Komutu

#define *önişlemci komutunun işlevi, metin düzenleyici programlardaki "bul - değiştir" (find - replace) özelliğine benzetilebilir. Bu komut kaynak kod içindeki bir yazıyı başka bir* yazı ile değiştirmek için kullanılır.

Önişlemci, *define sözcüğünden sonraki boşlukları atarak, boşluksuz ilk yazı kümesini elde* eder. Bu yazıya *STR1 diyelim. Daha sonra satır sonuna kadar olan tüm yazı kümesi elde* edilir. Buna da *STR2 diyelim. Önişlemci, kaynak kod içinde STR1 yazısı yerine STR2* yazısını yerleştirir:

#define SIZE 100

önişlemci komutuyla, önişlemci kaynak kod içinde gördüğü her bir *SIZE atomu yerine 100 atomunu yerleştirir. Derleme modülüne girecek kaynak programda, SIZE atomu artık* yer almaz.

#define önişlemci komutu kullanılarak çoğunlukla bir isim, sayısal bir değerle yer değiştirilir. Sayısal bir değerle değiştirilen isme, "simgesel değişmez" *(symbolic constant)* denir. Simgesel değişmezler nesne değildir. Derleme modülüne giren kaynak kodda, simgesel değişmezlerin yerini sayısal ifadeler almış olur. #define önişlemci komutuyla tanımlanan isimlere, "basit makro" (*simple macro) da denir.* Simgesel değişmezler, geleneksel olarak büyük harf ile isimlendirilir. Böylece kodu okuyanın değişkenlerle, simgesel değişmezleri ayırt edebilmesi sağlanır. Bilindiği gibi C dilinde, değişken isimlendirmelerinde ağırlıklı olarak küçük harfler kullanılır. Bir simgesel değişmez, başka bir simgesel değişmezin tanımlamasında kullanılabilir.

Örneğin:

```
#define MAX 100
#define MIN (MAX - 50)
```

Yer değiştirme işlemi, *STR1'in kaynak kod içinde bir atom halinde bulunması durumunda* yapılır:

```
#define SIZE 100
```

Bu tanımlamadan sonra kaynak kodda

```
size = MAX_SIZE;
printf("SIZE = %d\n", size);
```

gibi deyimlerin bulunduğunu düşünelim. Önişlemci bu deyimlerin hiçbirinde bir değişiklik yapmaz.

#define önişlemci komutu ile değişmezlere ve işleçlere ilişkin yer değiştirme işlemi yapılamaz.
Aşağıdaki *#define önişlemci komutları geçerli değildir:*

```
#define + -
#define 100 200
```

Simgesel değişmezler, C dilinin değişken isimlendirme kurallarına uygun olarak isimlendirilmelidir:

```
#define BÜYÜK 10
```

tanımlaması geçersizdir.
#define önişlemci komutunda dizgeler de kullanılabilir:

```
#define HATA_MESAJI "DOSYA  AÇILAMIYOR \n"
/***/
printf(HATA_MESAJI);
/***/
```

Simgesel değişmezler, yazılan kodun okunabilirliğini ve algılanabilirliğini artırır. Bazı değişmezlere isimlerin verilmesi, bu değişmezlerin ne amaçla kullanıldığı hakkında daha fazla bilgi verilebilir. Aşağıdaki örneğe bakalım:

```
#define PERSONEL_SAYISI 750
void foo()
{
/***/
if (x == PERSONEL_SAYISI)
/***/
}
```

Simgesel Değişmezlerin Tanımlanma Yerleri

#define komutu kaynak kodun herhangi bir yerinde kullanılabilir. Ancak tanımlandığı yerden kaynak kodun sonuna kadar olan bölge içinde etki gösterir. Önişlemci program doğrudan bilinirlik alanı kavramına sahip değildir. Bir bloğun başında tanımlanan bir simgesel değişmez yalnızca o bloğun içinde değil tanımlandığı yerden kaynak kodun sonuna kadar her yerde etkili olur.

Simgesel değişmezler bazen başlık dosyasının içinde bazen de kaynak dosyanın içinde tanımlanır.

Simgesel Değişmezlerin Kullanılmasında Sık Yapılan Hatalar

Tipk bir hata, simgesel değişmez tanımlamasında gereksiz yere '=' karakterini kullanmaktır:

#define N = 100

Bu durumda önişlemci *N gördüğü yere*

= 100

yazısını yapıştırır. Örneğin

int a[N];

gibi bir tanımlama yapıldığını düşünelim. Önişlemci bu tanımlamayı

a[= 100];

biçimine getirir ki bu da geçersizdir.

#define önişlemci komutu satırını yanlışlıkla ';' atomu ile sonlandırmak bir başka tipik hatadır.

#define N 100;

Bu durumda önişlemci *N gördüğü yere*

100;

yerleştirir.

int a[N];
tanımlaması
int a[100;];
haline gelir. Bu tanımlama geçersizdir. Bu tür hatalarda genellikle derleyici, simgesel değişmez kaç yerde kullanılmışsa o kadar hata iletisi verir.
Simgesel değişmezlerin tanımlanmasında dikkatli olunmalıdır. Önişlemci modülünün herhangi bir şekilde aritmetik işlem yapmadığı, yalnızca metinsel bir yer değiştirme yaptığı unutulmamalıdır:

```
#define MAX 10 + 20
int main()
{
int result;
result = MAX * 2;
printf("%d\n", result);
return 0;
}
```

Yukarıdaki örnekte *result değişkenine 60 değil 50 değeri atanır. Ancak önişlemci komutu*

```
#define MAX (10 + 20)
```

biçiminde yazılsaydı , *result değişkenine 60 değeri atanmış olurdu.*

switch DEYİMİ

switch deyimi bir tamsayı ifadesinin farklı değerleri için, farklı işlerin yapılması amacıyla kullanılır. *switch deyimi, özellikle else if merdivenlerine okunabilirlik yönünden bir* seçenek oluşturur.

Deyimin genel biçimi aşağıdaki gibidir:

```
switch (ifade) {
case ifade1 :
case ifade2 :
case ifade3 :
.......
case ifade_n:
default:
}
```

switch, case, ve default C dilinin anahtar sözcükleridir.

switch Deyiminin Yürütülmesi

switch ayracı içindeki ifadenin sayısal değeri hesaplanır. Bu sayısal değere eşit değerde bir *case ifadesi olup olmadığı yukarıdan aşağı doğru sınanır. Eğer böyle bir case ifadesi* bulunursa programın akışı o *case ifadesine geçirilir. Artık program buradan akarak ilerler. switch ayracı içindeki ifadenin sayısal değeri hiçbir case ifadesine eşit değilse, eğer varsa, default anahtar sözcüğünün bulunduğu kısma geçirilir.*

```
#include <stdio.h>
int main()
{
int a;
printf("bir sayi girin : ");
scanf("%d", &a);
switch (a) {
case 1: printf("bir\n");
case 2: printf("iki\n");
case 3: printf("üç\n");
case 4: printf("dört\n");
case 5: printf("beş\n");
}
return 0;
}
```

Yukarıdaki örnekte *scanf işlevi ile, klavyeden a değişkenine 1 değeri alınmış olsun. Bu* durumda programın ekran çıktısı şu şekilde olur:

bir
iki
üç
dört
beş

Eğer uygun *case ifadesi bulunduğunda yalnızca bu ifadeye ilişkin deyim(ler)in* yürütülmesi istenirse *break deyiminden faydalanılır. break deyiminin kullanılmasıyla,* döngülerden olduğu gibi *switch deyiminden de çıkılır. Daha önce verilen örneğe break* deyimleri ekleniyor:

```
#include <stdio.h>
int main()
{
int a;
printf("bir sayi girin: ");
scanf("%d", &a);
switch (a) {
case 1 : printf("bir\n"); break;
case 2 : printf("iki\n"); break;
case 3 : printf("üç\n"); break;
case 4 : printf("dört\n"); break;
case 5 : printf("beş\n");
}
return 0;
}
```

case ifadelerini izleyen ":" atomundan sonra istenilen sayıda deyim olabilir. Bir case ifadesini birden fazla deyimin izlemesi durumunda bu deyimlerin bloklanmasına gerek yoktur. Yani bir *case ifadesini izleyen tüm deyimler, bir blok içindeymiş gibi ele alınır. case ifadelerinin belirli bir sırayı izlemesi gibi bir zorunluluk yoktur.*

default case

default bir anahtar sözcüktür. switch deyimi gövdesine yerleştirilen default anahtar sözcüğünü *':' atomu izler. Oluşturulan bu case'e default case denir.*

Eşdeğer bir *case ifadesi bulunamazsa programın akışı default case içine girer.*

Daha önce yazılan *switch deyimine default case ekleniyor.*

```
#include <stdio.h>
int main()
{
int a;
printf("bir sayi girin: ");
scanf("%d", &a);
switch (a) {
case 1 : printf("bir\n"); break;
case 2 : printf("iki\n"); break;
case 3 : printf("üç\n"); break;
case 4 : printf("dört\n"); break;
case 5 : printf(“beş\n"); break;
default: printf("hiçbiri\n");
}
return 0;
}
```

case ifadelerinin, tamsayı türünden (integral types) değişmez ifadesi olması gerekir. Bilindiği gibi değişmez ifadeleri, derleme aşamasında derleyici tarafından net sayısal değerlere dönüştürülebilir:

case 1 + 3: **/* Geçerli */**

mümkün çünkü *1 + 3 değişmez ifadesi ama ,*

case x + 5: **/* Geçersiz */**

çünkü değişmez ifadesi değil. Derleyici, derleme aşamasında sayısal bir değer hesaplayamaz.

case 'a' :

Yukarıdaki *case ifadesi geçerlidir. 'a' bir karakter değişmezidir. case ifadesi tamsayı* türünden bir değişmez ifadesidir.

case 3.5 :

Yukarıdaki *case ifadesi geçersizdir. 3.5 bir gerçek sayı değişmezidir.*

switch kontrol deyimi yerine bir else if merdiveni yazılabilir. Yani switch deyimi olmasaydı, yapılmak istenen iş, bir *else if merdiveni ile de yapılabilirdi. Ancak bazı* durumlarda *else if merdiveni yerine switch deyimi kullanmak okunabilirliği artırır.*

```
if (a == 1)
deyim1;
else if (a == 2)
deyim2;
else if (a == 3)
deyim3;
else if (a == 4)
deyim4;
else
deyim5;
```

Yukarıdaki *else if merdiveni ile aşağıdaki switch deyimi işlevsel olarak eşdeğerdir:*

```
switch (a) {
case 1 : deyim1; break;
case 2 : deyim2; break;
case 3 : deyim3; break;
case 4 : deyim4; break;
default: deyim5;
}
```

Her *switch deyiminin yerine aynı işi görecek şekilde bir else if merdiveni yazılabilir ama her else if merdiveni bir switch deyimiyle karşılanamaz*. *switch ayracı içindeki ifadenin bir* tamsayı türünden olması zorunludur. *case ifadeleri de tamsayı türlerinden değişmez* ifadesi olmak zorundadır. *switch deyimi, tamsayı türünden bir ifadenin değerinin değişik* tamsayı değerlerine eşitliğinin sınanması ve eşitlik durumunda farklı işlerin yapılması için kullanılır. Oysa *else if merdiveninde her türlü karşılaştırma söz konusu olabilir. Örnek:*

```
if (x > 20)
m = 5;
else if (x > 30 && x < 55)
m = 3;
else if (x > 70 && x < 90)
m = 7;
else
m = 2;
```

Yukarıdaki *else if merdiveninin yerine bir switch deyimi yazılamaz.*

Birden fazla *case ifadesi için aynı işlemlerin yapılması şöyle sağlanabilir.*

```
case 1:
case 2:
case 3:
deyim1;
deyim2;
break;
case 4:
```

Bunu yapmanın daha kısa bir yolu yoktur. Bazı programcılar kaynak kodun yerleşimini aşağıdaki gibi düzenlerler:

```
case 1: case 2: case 3: case 4: case 5:
deyim1; deyim2;
```

```
void print_season(int month)
{
if (month == 12 || month == 1 || month == 2)
printf("winter");
else if (month == 3 || month == 4 || month == 5)
printf("spring");
else if (month == 6 || month == 7 || month == 8)
printf("summer");
else if (month == 9 || month == 10 || month == 11)
printf("autumn");
}
```

Aşağıdaki programı önce inceleyin, sonra derleyerek çalıştırın:

```
void print_season(int month)
{
switch (month) {
case 12:
case 1 :
case 2 : printf("winter"); break;
case 3 :
case 4 :
case 5 : printf("spring"); break;
case 6 :
case 7 :
case 8 : printf("summer"); break;
case 9 :
case 10:
case 11: printf("autumn");
}
}
```

Simgesel değişmezler, derleme işleminden önce önişlemci tarafından değiştirileceği için, *case ifadelerinde yer alabilir:*

```
#define TRUE 1
#define FALSE 0
#define UNDEFINED 2
case TRUE :
case FALSE :
case UNDEFINED :
```

Yukarıdaki *case ifadeleri geçerlidir.*

case ifadeleri olarak karakter değişmezleri de kullanılabilir:

```
#include <stdio.h>
int main()
{
switch (getchar()) {
case '0': printf("sıfır\n"); break;
case '1': printf("bir\n"); break;
case '2': printf("iki\n"); break;
case '3': printf("üç\n"); break;
case '4': printf("dört\n"); break;
case '5': printf("beş\n"); break;
default : printf("gecersiz!\n");
}
return 0;
}
```

Bir *switch deyiminde aynı sayısal değere sahip birden fazla case ifadesi olamaz. Bu* durum derleme zamanında hata oluşturur.
switch deyimi, başka bir switch deyiminin ya da bir döngü deyiminin gövdesini oluşturabilir:

```
#include <stdio.h>
#include <conio.h>
#define ESC 0X1B
int main()
{
int ch;
while ((ch = getch()) != ESC)
switch (rand() % 7 + 1) {
case 1: printf("Pazartesi\n"); break;
case 2: printf("Sali\n"); break;
case 3: printf("Carsamba\n"); break;
case 4: printf("Persembe\n"); break;
case 5: printf("Cuma\n"); break;
case 6: printf("Cumartesi\n"); break;
case 7: printf("Pazar\n");
}
return 0;
}
```

rand İşlevi

Standart *rand işlevi rastgele sayı üretir. Bu işlevin bildirimi aşağıdaki gibidir:*

int rand(void);

C standartları *rand işlevinin rastgele sayı üretimi konusunda kullanacağı algoritma ya da* teknik üzerinde bir koşul koymamıştır. Bu konu derleyiciyi yazanların seçimine bağlı(*implementation dependent) bırakılmıştır. rand işlevinin bildirimi, standart bir başlık* dosyası olan *stdlib.h içindedir. Bu yüzden rand işlevinin çağrılması durumunda bu başlık* dosyası *"include" önişlemci komutuyla kaynak koda eklenmelidir.*

#include <stdlib.h>

rand işlevi her çağrıldığında [0, RAND_MAX] aralığında rastgele bir tamsayı değerini geri döndürür. *RAND_MAX stdlib.h başlık dosyası içinde tanımlanan bir simgesel değişmezdir.* C standartları bu simgesel değişmezin en az *32767 değerinde olmasını şart koşmaktadır.* Derleyicilerin hemen hepsi *RAND_MAX simgesel değişmezini 32767 olarak, yani 2 byte* işaretli *int türünün en büyük değeri olarak tanımlar:*

#define RAND_MAX 32767

Aşağıdaki program parçasında, *0 ile RAND_MAX arasında 10 adet rastgele sayı üretilerek* ekrana yazdırılıyor. Programı derleyerek çalıştırın:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int k;
for (k = 0; k < 10; ++k)
printf("%d ", rand());
return 0;
}
```

Aşağıdaki programı inceleyin. Programda *display_date isimli bir işlev* tanımlanıyor. İşlev gün, ay ve yıl değeri olarak aldığı bir tarih bilgisini İngilizce olarak aşağıdaki formatta ekrana yazdırıyor:

Örnek:

5th Jan 1998

```
include <stdio.h>
void display_date(int day, int month, int year)
{
printf("%d", day);
switch (day) {
case 1 :
case 21 :
case 31 : printf("st "); break;
case 2 :
case 22 : printf("nd "); break;
case 3 :
case 23 : printf("rd "); break;
default : printf("th ");
}
switch (month) {
case 1 : printf("Jan "); break;
case 2 : printf("Feb "); break;
case 3 : printf("Mar "); break;
case 4 : printf("Apr "); break;
case 5 : printf("May "); break;
case 6 : printf("Jun "); break;
case 7 : printf("Jul "); break;
case 8 : printf("Aug "); break;
case 9 : printf("Sep "); break;
case 10: printf("Oct "); break;
case 11: printf("Nov "); break;
case 12: printf("Dec ");
}
printf("%d", year);
}
```

```
int main()
{
int day, month, year;
int n = 20;
while (n-- > 0) {
printf("gun ay yil olarak bir tarih girin : ");
scanf("%d%d%d", &day, &month, &year);
display_date(day, month, year);
putchar('\n');
}
return 0;
}
```

İşlevin tanımında iki ayrı *switch deyimi kullanılıyor. İlk switch deyimiyle, gün değerini* izleyen *(th, st, nd, rd) sonekleri yazdırılırken, ikinci switch deyimiyle, aylara ilişkin* kısaltmalar (J*an, Feb. Mar.) yazdırılıyor.*
case ifadelerini izleyen deyimlerden biri break deyimi olmak zorunda değildir. Bazı durumlarda *break deyimi özellikle kullanılmaz, uygun bir case ifadesi bulunduğunda daha* aşağıdaki *case lerin içindeki deyimlerin de yapılması özellikle istenir. Aşağıdaki programı* derleyerek çalıştırın:

```
#include <stdio.h>
int isleap(int y)
{
return y % 4 == 0 && (y % 100 != 0 || y % 400 == 0);
}
int day_of_year(int day, int month, int year)
{
int sum = day;
switch (month - 1) {
case 11: sum += 30;
case 10: sum += 31;
case 9 : sum += 30;
case 8 : sum += 31;
case 7 : sum += 31;
case 6 : sum += 30;
case 5 : sum += 31;
case 4 : sum += 30;
case 3 : sum += 31;
case 2 : sum += 28 + isleap(year);
case 1 : sum += 31;
}
return sum;
}
```

```
int main()
{
int day, month, year;
int n = 5;
while (n-- > 0) {
printf("gun ay yil olarak bir tarih girin : ");
scanf("%d%d%d", &day, &month, &year);
printf("%d yilinin %d. gunudur!\n", year, day_of_year(day, month,
year));
}
return 0;
}
```

day_of_year işlevi dışarıdan gün, ay ve yıl değeri olarak gelen tarih bilgisinin ilgili yılın kaçıncı günü olduğunu hesaplayarak bu değerle geri dönüyor.

goto DEYİMİ

Diğer programlama dillerinde olduğu gibi C dilinde de programın akışı, bir koşula bağlı olmaksızın kaynak kod içinde başka bir noktaya yönlendirilebilir. Bu, C dilinde *goto* deyimi ile yapılır:

goto deyiminin genel sözdizimi aşağıdaki gibidir:

<goto etiket;>

....

<etiket:>

<deyim;>

goto, C dilinin 32 anahtar sözcüğünden biridir. Etiket (label), programcının verdiği bir isimdir. Şüphesiz isimlendirme kurallarına uygun olarak seçilmelidir. Programın akışı, bu etiketin yerleştirilmiş olduğu yere yönlendirilir. Etiket, *goto anahtar sözcüğünün* kullanıldığı işlev içinde herhangi bir yere yerleştirilebilir. Etiket isminden sonra *':' atomu* yer almak zorundadır. Etiketi izleyen deyim de *goto kontrol deyiminin sözdiziminin bir* parçasıdır. Etiketten sonra bir deyimin yer almaması bir sözdizim hatasıdır.

Etiketin *goto anahtar sözcüğünden daha sonraki bir kaynak kod noktasına yerleştirilmesi* zorunluluğu yoktur. Etiket *goto anahtar sözcüğünden önce de tanımlanmış olabilir:*

```
int main()
{
/***/
goto GIT;
/***/
GIT:
printf("goto deyimi ile buraya gelindi\n");
return 0;
}
```

Yukarıdaki programda, etiket goto anahtar sözcüğünden daha sonra yer alıyor.

```
int main()
{
GIT:
printf("goto deyimi ile gelinecek nokta\n");
/***/
goto GIT;
/***/
return 0;
}
```

Yukarıdaki programda, etiket goto anahtar sözcüğünden daha önce yer alıyor.

goto etiketleri, geleneksel olarak büyük harf ile, birinci sütuna dayalı olarak yazılır. Böylece kaynak kod içinde daha fazla dikkat çekerler.
goto etiketleri bir işlev içinde, bir deyimden önce herhangi bir yere yerleştirilebilir. Yani etiket, aynı işlev içinde bulunmak koşuluyla, *goto anahtar sözcüğünün yukarısına ya da* aşağısına yerleştirilebilir. Bu özelliğiyle *goto etiketleri, yeni bir bilinirlik alanı kuralı* oluşturur.
Yapısal programlama tekniğinde *goto deyiminin kullanılması önerilmez. Çünkü goto* deyiminin kullanılması bir takım sakıncalar doğurur:
1. goto deyimi programların okunabilirliğini bozar. Kodu okuyan kişi goto deyimiyle karşılaştığında işlevin içinde etiketi arayıp bulmak zorunda kalır ve programı bu noktadan okumayı sürdürür.
2. *goto deyimlerinin kullanıldığı bir programda bir değişiklik yapılması ya da programın,* yapılacak eklemelerle, geliştirilmeye çalışılması daha zor olur. Programın herhangi bir yerinde bir değişiklik yapılması durumunda, eğer program içinde başka yerlerden değişikliğin yapıldığı yere *goto deyimleri ile sıçrama yapılmış ise, bu noktalarda da bir* değişiklik yapılması gerekebilir. Yani *goto deyimi program parçalarının birbirine olan* bağımlılığını artırır, bu da genel olarak istenen bir şey değildir.

Bu olumsuzluklara karşın, bazı durumlarda *goto deyiminin kullanılması programın* okunabilirliğini bozmak bir yana, diğer seçeneklere göre daha okunabilir bir yapının oluşmasına yardımcı olur:
İçiçe birden fazla döngü varsa, ve içteki döngülerden birindeyken, yalnızca bu döngüden değil, bütün döngülerden birden çıkılmak isteniyorsa *goto deyimi kullanılmalıdır.*

```
#include <stdio.h>
int test_func(int val);
int main()
{
int i, j, k;
for (i = 0; i < 100; ++i) {
for (j = 0; j < 100; ++j) {
for (k = 0; k < 20; ++k) {
/*...*/
if (!test_func(k))
goto BREAK;
/*...*/
}
}
}
BREAK:
printf("döngü dışındaki ilk deyim\n");
return 0;
}
```

TO BE CONTINUED...