ALT BAŞLIKLAR

2. Algoritmalarda Kullanılan Temel Operatörler
   2.1. Operatör Nedir?
   2.2. Matematiksel Operatörler
      2.2.1. "+" Toplama Operatörü
      2.2.2. "-" Çıkartma Operatörü
      2.2.3. "*" Çarpma Operatörü
      2.2.4. "/" Bölme Operatörü
      2.2.5. "^" Üs Operatörü
   2.3. Karşılaştırma Operatörleri
      2.3.1. ">" Büyüktür Operatörü
      2.3.2. "<" Küçüktür Operatörü
      2.3.3. "==" Eşittir Operatörü
      2.3.4. ">=" Büyük Eşit Operatörü
      2.3.5. "<=" Küçük Eşit Operatörü
      2.3.6. "<>" Eşit Değildir Operatörü
   2.4. Mantıksal Operatörler
      2.4.1. "AND" Ve Operatörü
      2.4.2. "OR" Veya Operatörü
      2.4.3. "NOT" Değil Operatörü
      2.4.4. Mantıksal Operatörlerin Doğruluk Tablosu
      2.4.5. Mantıksal Operatörlerin Elektrik Devreleri İle İfade Edilmesi
   2.5. "=" Atama Operatörü
   2.6. Operatör Önceliği
   2.7. Algoritma Örnekleri
      2.7.1. Örnek 1
      2.7.2. Örnek 2

ÜNİTE HAKKINDA

Günlük hayatımızda kullandığımız dillerde; sesler heceleri, heceler sözcükleri, sözcükler cümleleri oluşturmaktadır. Bu birleştirmeler sırasında edatlar, bağlaçlar gibi bir takım ara yapılar kullanılır. Sonuçta ortaya çıkan cümlelerle de insanlar kendi aralarında iletişim kurmaktadırlar. Programlama dilleriyle de "insanlar ile bilgisayarlar" arasında bir iletişim kurulmaktadır.

Programlama dillerinin de kendilerine has cümleleri vardır. Programlama dillerinin cümlelerine "deyimler" denmektedir. Aynen günlük hayatımızda ki dillerde olduğu gibi programlama dillerinin cümleleri (deyimleri), bir takım ara yapılarla daha alt ifadelerin birleştirilmesinden oluşmuştur.

Programlama dillerindeki bu ara yapılara ise "operatörler" denmektedir. Operatörler programcılar için önemlidir. Çünkü yapmak istediğimiz daha doğrusu bilgisayara yaptırmak istediğimiz işlerin hemen hemen her aşamasında operatörlerden yararlanmaktayızdır. Bu toplama-çıkarma gibi bir aritmetik işlemde olabilir ya da eşittir-eşit değildir gibi bir karşılaştırma işlemi de olabilir.

Programlama dillerinin güçleri, operatörlerinin zenginliği ile de bağlantılıdır. Bütün programlama dillerinin güçleri aynı olmadığı gibi dolayısıyla bütün programlama dillerinde kullanılan operatörlerde birbirleriyle aynı değildir. Fakat kullanılan bazı temel operatörler vardır ki programlama dillerinin hemen hemen hepsinde olmasını bekleriz. Örneğin bir toplama işlemi, bütün programlama dillerinde bulunmaktadır.

Bu ünitede de henüz programlama dillerinden bağımsız olarak problemlerimizin algoritmalarını kurmaya (çözüm aşamalarını geliştirmeye) çalıştığımız için, herhangi bir programlama dilinin operatörleri üzerinde durmaktansa, algoritmalarda kullanılan temel bazı operatörler işlenmeye çalışılacaktır.

İleriki ünitelerde ise özel olarak Pascal programlama dilinde kullanılan operatörlerden bahsedilecektir.

ÖĞRENME HEDEFLERİ

Bu ünite bitiminde;

- Operatör nedir? Tanımlayabilecek,
- Temel operatörleri sayabilecek,
- Temel operatörlerin işlevlerini söyleyebilecek,
- Mantıksal operatörlerin doğruluk tablosunu çıkartabilecek,
- Mantıksal operatörleri elektrik devreleri ile ifade edebilecek,
- Temel operatörler arasındaki işlem önceliğinin farkında olabilecek ve
- Temel operatörleri algoritmalarınızda kullanabileceksiniz.

ÜNİTEYİ ÇALIŞIRKEN

Algoritmalarda Kullanılan Temel Operatörler Ünitesinde bir dizi operatör ve bunların kullanım alanları işlenecektir. Değişik kaynaklardan, işlenecek olan bu operatörlere ait kullanım örneklerini inceleyip, kesişimler oluşturması açısından ders notları ile karşılaştırma yoluna gidebilirsiniz.

Ayrıca operatörler konusunda şimdilik, programlama dilinden bağımsız olarak hareket etmemize rağmen, işlenecek olan operatörlerin Pascal Programlama Dilindeki karşılıklarını şimdiden araştırabilirsiniz.

Aralarındaki farkları da not etmeniz iyi olur. Örneğin atama operatörü. Algoritmalarda kullanılan atama operatörü aynen Pascal programlama dilinde de kullanılıyor mu? Yoksa küçük değişiklikler olabiliyor mu? Gibi…

ANA METİN

2. Operatörler

2.1. Operatör Nedir?

✓Operatörler önceden tanımlanmış birtakım matematiksel ya da mantıksal işlemleri yapmak için kullanılan özel karakter ya da karakterler topluluğudur. (Algan, 2008)

İşlem basamaklarımızın oluşturulmasında yeri geldikçe bazı temel operatörlerden yararlanmamız gerecektir. Örneğin iki sayının toplamını almak için, bölümden kalanını bulmak için veya üssünü hesaplamak için… Ya da iki sayıdan büyük olana karar vermek, aralarındaki eşitliği bulmak için gibi…

Operatörler gerekli işleri yapmaları için birtakım malzemeye ihtiyaç duyarlar. Örneğin "+" operatörünün toplama yapabilmesi için iki tane sayıya ihtiyacı olduğu gibi. Bu sayılara operand denilmektedir. (Algan, 2008)

Her operatörün farklı sayıda operandları olabilmektedir.

✓Operand, ilgili operatör tarafından işleme katılan değerlerdir.

Programlama dillerinin destekledikleri operatörlerin değişiklik göstermesine rağmen programlama dillerinden bağımsız olarak hazırlayacağımız algoritmalarımız da kullanabileceğimiz temel operatörleri işlevlerine göre aşağıdaki gibi gruplandırabiliriz.

2.2. Matematiksel Operatörler

Operandları arasında temel bazı matematiksel işlemleri yerine getirebilen operatörlerdir.

2.2.1. "+" Toplama operatörü: Matematikteki bildiğimiz toplama işlemidir. Normalde iki operand arasında toplama yapar (işleme katılan operandlar sayısal türden iseler)

*3+5 ➜ 8, x=4 => x+1 ➜ 5 gibi…

!İşleme katılan operandlar alfasayısal (sayısal olmayan, yazı türünden) tipli değerler ise "+" operatörü birleştirme işlemi yapar.

* "Ali" + "Veli" ➜ "AliVeli" gibi…

2.2.2. "-" Çıkartma operatörü: Matematikteki bildiğimiz çıkartma işlemidir İki operand arasında çıkartma yapar.

*5-3 ➜ 2, x=4 => x-1 ➜ 3

2.2.3. "*" Çarpma operatörü: Matematikteki bildiğimiz çarpma işlemidir İki sayı arasında çarpma yapar.

*5*3 ➔ 15, z=2 => z*3 ➔ 6

!Bilgisayar dünyasında kullanılan programların çoğunda; (Programlama Dilleri, Hesap Makinesi, Excel vb.) çarpma işlemi matematikteki gibi "." veya "x" ile ifade edilmemektedir. Onların yerine "*" kullanılmaktadır.

2.2.4. "/" Bölme operatörü: Matematikteki bildiğimiz bölme işlemidir İki sayı arasında bölme yapar.

*12/3 ➔ 4, x=4 => 12/x ➔ 3

!Bölme işleminde dikkat etmemiz gereken nokta, ikinci sayının bir şekilde "0 (Sıfır)" olmaması gerektiğidir. Çünkü "sayı/0" ifadesi anlamsız bir sonuç üreteceğinden programımız da kararsız duruma düşebilir.

2.2.5. "^" Üs operatörü: Üs alma işlemidir. Taban bu operatörün soluna yazılırken üs ise sağ tarafına yazılır.

*2^3 ➔ 8

!Bir programcının 2'nin kuvvetlerini ezbere bilmesi gerekmez fakat bir süre sonra "2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768…" diyerek seri bir şekilde sayabildiğinizi göreceksiniz. Bu da bilgisayar mimarisinin 2'lik sayılar üzerine kurulmuş ve dolayısıyla da bu sayılara sıkça işimizin düşüyor olmasından kaynaklanmaktadır.

2.3. Karşılaştırma Operatörleri

Operandları arasında büyüklük, küçüklük, eşitlik gibi karşılaştırma işlemleri yapabilen operatörlerdir.

!Karşılaştırma işlemlerinde sonuç, ya doğrudur ya da değildir. Az doğru ya da çok yanlış gibi bir şey söz konusu değildir. Aslına bakılırsa bilgisayar dediğimiz bu çok akıllı gibi görünen alet hep bu mantık üzerine inşa edilmiştir. Doğru ya da yanlış, evet ya da hayır, "1" ya da "0" gibi…

Bilgisayar sistemlerinde ikilik sayı sisteminin kullanılıyor olması da buradan gelmektedir. Sonuçta bilginin ifade edilmesi ve bir yerden bir yere de taşınması gerekecektir. Bilgisayar sistemlerinde bu işlem elektrik vasıtasıyla yapılmaktadır. Yine yukarıdaki mantıkla kablonun ucunda elektrik (+5 Volt) varsa "1" yoksa "0" kabul edilmesi gibi. Üçüncü bir ihtimal kullanılmamaktadır. Bu iki sayı da ("1" ve "0") ikilik sayı sistemini oluşturmaktadır.

2.3.1. “>” Büyüktür operatörü: Operandları arasında ki büyüklük ilişkisini yakalamaya çalışır.

! Yakalamaktan kasıt doğru sonucunu üretmesidir. Eğer ki solundaki operandın sağındakinden büyük olduğunu tespit ederse “evet, doğrudur” sonucunu üretir, aksi halde “hayır, yanlıştır” sonucunu üretir.

* 5>3 ➔ Evet, doğrudur (5, 3’den büyüktür), x=4 ve y=8 => x>y ➔ Hayır, yanlıştır (bu değerlere göre x, y’den büyük değildir) gibi…

2.3.2. “<” Küçüktür operatörü: Operandları arasında ki küçüklük ilişkisini yakalamaya çalışır.

* 4<5 ➔ Doğru, 10<5➔ Yanlış

2.3.3. “==” Eşittir operatörü: Operandları arasında ki eşitlik ilişkisini yakalamaya çalışır.

* 5==5 ➔ Doğru, 3==6➔ Yanlış

!Operatörler programlama dillerinde farklı sembollerle gösterilebilmektedir. İleriki konularımızda göreceğimiz Pascal programlama dilindeki “eşittir” operatörü de mantık olarak yukarıda açıklanan “==” eşittir operatörüyle aynı işi yapmasına rağmen “=” tek eşittir işaretiyle gösterilmektedir. Bu durum diğer bazı operatörlerde de geçerlidir ve Pascal programlama dilindeki operatörlere yeri geldiğinde değinilecektir. Burada ki amacımız programlama dillerinden bağımsız olarak operatörlerin kullanım amaçlarını kavramak olmalıdır.

2.3.4. “>=” Büyük Eşit operatörü: Operandları arasında ki büyüklük veya eşitlik ilişkisini yakalamaya çalışır.

* 5>=4 ➔ Doğru, 5>=10➔ Yanlış, 8>=8➔Doğru

2.3.5. “<=” Küçük Eşit operatörü: Operandları arasında ki küçüklük veya eşitlik ilişkisini yakalamaya çalışır.

* 4<=5 ➔ Doğru, 10<=5➔ Yanlış, 8<=8➔Doğru

2.3.6. “<>” Eşit Değildir operatörü: Operandları arasında ki eşit olmama durumunu yakalamaya çalışır.

*5<>4 ➔ Doğru (Evet, doğru. 5, 4'e eşit değildir gibi…), 3<>3 ➔Yanlış

2.4. Mantıksal Operatörler

Aslında çok ilginçtir, günlük hayatta bile farkında olmadan sıkça kullandığımız bu operatörler bilgisayar dünyasında da sıklıkla kullanılmaktadırlar. Operandlarını, adı üzerinde mantıksal olarak birbirine bağlarlar. Kullanılan işlemler "ve", "veya" ya da "değil"i gibi mantıksal işlemler olabilir.

!Mantıksal operatörlerin operandları da mantıksal olarak işlenmeye uygun olmalıdır. Bu şu anlama gelmektedir: Operandlar, "doğru" ya da "yanlış" değerlerinden birisi olmalıdır. Üçüncü bir ihtimalin mantıksal operatörler tarafından işlenmesi mümkün değildir.

2.4.1. "AND" Ve operatörü: Birbirine bağladığı operandların tümünün "doğru" olması durumunu yakalamaya çalışır. Ancak bu durumda sonuç "doğru" olmaktadır aksi halde (yani operandların birisi bile yanlış ise) sonuç "yanlış" olarak üretilmektedir.

Diğer bir ifadeyle "ve operatörü" bütün durumların aynı anda gerçekleştirilmiş (sağlanmış) olmasını gerektirir.

*(4==4) AND (2>1) ➔ Doğru, Çünkü hem "4, 4'e eşittir hem de 2, 1'den büyüktür". Yani bütün durumlar gerçeklenmiştir.

*(4==4) AND (2>1) AND (5<3) ➔ Yanlış, Çünkü "Tamam 4, 4'e eşittir, 2'de 1'den büyüktür fakat 5, 3'ten küçük olmadığı için sonuç yanlıştır".

! Ve operatöründe sonucun doğru çıkması ancak bütün durumların sağlanması ile mümkündür. Günlük hayatta da aslında bu ifadeyi kullanmaktayız. Örneğin bir grup öğrenci arasından "sarışın ve gözlüklü olanları seçmeye çalıştığınızı düşünün. Seçtiğiniz öğrencilerin aynı anda iki özelliği taşımasını istediğiniz içindir ki bu iki özelliği "ve operatörüyle birbirine bağlamışsınızdır". Bu durumda seçilen öğrenciler hem sarışındırlar hem de gözlük kullanmaktadırlar. Sarışın olup ta gözlük kullanmayan ya da gözlük kullanıp ta sarışın olmayan öğrenciler bu ifadeye göre seçilememektedir. Çünkü bütün durumların aynı anda gerçeklenmesi istenmektedir.

!Mantıksal operatörlerin sonuçları da "doğru" ya da "yanlıştır". Üçüncü bir ihtimal söz konusu değildir.

2.4.2. "OR" Veya operatörü: Birbirine bağladığı operandların en az bir tanesinin "doğru" olması durumunu yakalamaya çalışır. Operandlardan bir veya birden fazlası ya da hepsinin doğru olması durumunda veya operatörü "doğru" sonucunu üretir.

Diğer bir ifadeyle "veya operatörü"nde "ve" operatörünün aksine "doğru" sonucunun üretilebilmesi için şartlardan bir tanesinin bile gerçekleştirilmiş (sağlanmış) olması yeterlidir.

*(4==4) OR (1>5) ➔ Doğru, Çünkü "1'in 5'ten büyük olmamasına rağmen sonucun doğru çıkması için 4'ün 4'e eşit olması yeterli olmuştur".

*(4==4) OR (2>1) ➔ Doğru, Çünkü hem "4, 4'e eşittir hem de 2, 1'den büyüktür". Aslında bir tanesinin bile doğru olması yeterliyken bütün şartlar sağlanmıştır.

2.4.3. "NOT" Değil operatörü: Operandının durumunu tersine çevirir. Yani durum doğruysa sonuç "yanlış" olarak, yanlış ise de sonucun doğru olarak üretilmesini sağlar.

*NOT(1=1) ➔ Yanlış, Çünkü "1'in 1'e eşit olmasına rağmen, sonuç değil operatörüyle tersine çevrilmiştir".

2.4.4. Mantıksal Operatörlerin Doğruluk Tablosu

Mantıksal operatörlerde işleme katılan operandların ve işlem sonucunun "doğru" ya da "yanlış" değerlerinden birisi olması gerektiğine değinilmişti. Bilgisayar dünyasında doğru ve yanlışlar, "1" ve "0" lar ile de ifade edilmekteydi. Şimdi 1'leri "doğru" ve 0'ları da "yanlış" kabul ederek mantıksal operatörlerimizin olabilecek tüm ihtimallerde hangi sonuçları ürettiğini tablolar üzerinde görmeye çalışalım.

Operand 1 (p)	Operand 2 (q)	(p) **AND** (q)
0	0	0
0	1	0
1	0	0
1	1	1

"AND" Operatörünün Doğruluk Tablosu

Operand 1 (p)	Operand 2 (q)	(p) **OR** (q)
0	0	0
0	1	1
1	0	1
1	1	1

"OR" Operatörünün Doğruluk Tablosu

Operand 1 (p)	**NOT** (p)

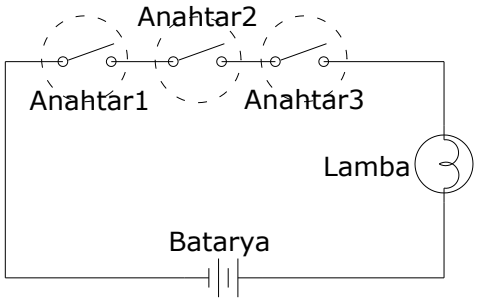
0	1
1	0

“NOT” Operatörünün Doğruluk Tablosu

✔Bu şekilde, mantıksal operatörlerin operandlarının alabileceği tüm ihtimalleri ve oluşabilecek tüm sonuçları tek bir tablo üzerinde gösterme işlemine “Doğruluk Tablosunu Çıkarma” denir.

2.4.5. Mantıksal Operatörlerin Elektrik Devreleri İle İfade Edilmesi

Mantıksal operatörlerin çalışma mantıkları basit elektrik devrelerinin çalışma mantıklarıyla çok daha rahat anlaşılabilmektedir. Bu amaçla hazırlanmış aşağıdaki devreleri inceleyin.

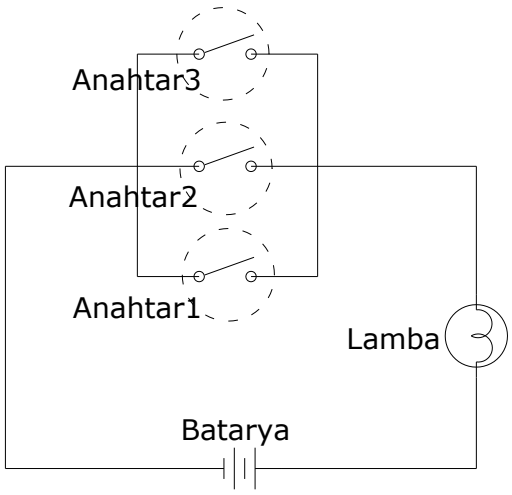


“AND” Operatörünün Elektrik Devreleriyle Açıklanması

Yukarıdaki devrede bataryadan alınan güçle lambanın yanabilmesi için devrenin tamamlanması gerektiği açıktır. Bunun için de anahtarların kapalı olması gerekmektedir.

Burada üzerinde düşünmemiz gereken nokta, anahtarlardan sadece bir tanesinin veya iki tanesinin kapalı olması durumunda lambanın yanıp yanmayacağıdır. Bu sorunun cevabı tabiî ki hayır, yanmaz şeklinde olmalıdır. Çünkü lambanın yanması için birbirine seri bir şekilde bağlanan anahtarların üçünün birden kapalı durumda olması gerekmektedir. Ancak bu durumda devre tamamlandığından lamba yanabilmektedir.

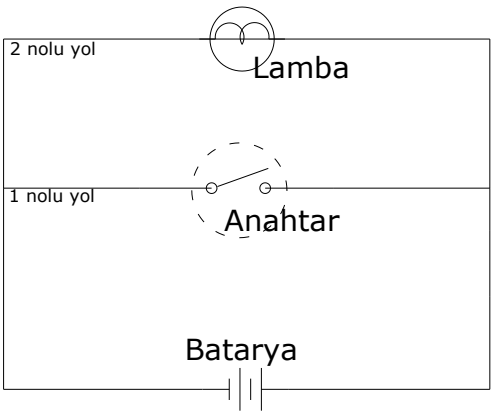
Devrenin bu çalışma mantığı da bizi “ve operatörünün” çalışma mantığına götürmektedir. Zira “ve operatörü”de doğru sonucunu üretebilmek için (lambanın yanması) tüm şartların sağlanmış olmasını istemekteydi. Burada anahtarları “ve operatörü”nün operandları yani sağlanması gereken durumlar olarak düşündüğümüzde, lambanın yanması için tüm anahtarların kapalı olması yani tüm şartların gerçeklenmesi gerektiği görülmektedir.



“OR” Operatörünün Elektrik Devreleriyle Açıklanması

Yukarıdaki devrede de bataryadan alınan güçle lambanın yanabilmesi için devrenin tamamlanması gerektiği açıktır. Yalnız burada devrenin tamamlanabilmesi için bütün anahtarların kapalı olmasına gerek yoktur. Yalnızca bir tanesinin kapalı olması durumunda bile devre tamamlandığı için lamba yanabilmektedir.

Yine bu devrenin çalışma mantığı da bizi “veya operatörünün” çalışma mantığına götürmektedir. “Veya operatörü”nde de sonucun doğru çıkması için şartlardan (durumlardan) en az bir tanesinin sağlanması yeterli olmaktaydı.



“NOT” Operatörünün Elektrik Devreleriyle Açıklanması

“Değil operatörünün” çalışması da yukarıdaki gibi bir elektrik devresiyle açıklanabilmektedir. Devrede aynı şekilde lambanın yanmasını, sonucun “doğru” olması olarak kabul edersek; kendimize şu soruyu sormalıyız:

“Lamba anahtar kapalıyken mi yanar yoksa açıkken mi?”

Lamba anahtar açıkken yanar, çünkü anahtar açıkken devre zaten lamba üzerinden tamamlanmaktadır(2 nolu yol üzerinden). Oysa ki anahtar kapatılırsa bu defa devre kısa yolu tercih edeceğinden (1 nolu yolu) anahtar üzerinden tamamlanır ve lamba sönmüş olur. Dolayısıyla anahtar açıkken lamba yanar, anahtar kapalıyken de lamba söner.

Buda, tam da “değil operatörünün” yaptığı iştir. Zira değil operatörünün kontrol ettiği operand (şart ya da durum) yanlış ise sonucu “doğru” doğruysa da sonucu “yanlış” olmaktaydı.

2.5. “=” Atama Operatörü

Önceki ünitede ki bir örnekte değinilen bu operatör, matematikteki “=” eşittir işlemine benzese de programcılıkta çok farklı bir şekilde kullanılır. Matematikte “x = (x+1)” işlemi imkansızdır. Çünkü eşittirin sağındaki değer solundaki değere eşit değildir. Oysaki programcılıkta en çok kullanacağımız ifadelerdendir.

Atama operatörü, programcılıkta atama işlemleri için kullanılmaktadır. Atama işlemi; sağdaki ifadenin hesaplanarak soldaki değişkene aktarılması şeklinde gerçekleşir. Yukarıdaki ifadede geçen “=” işaretide programcılıkta “eşittir” işlemi şeklinde değil de “atama” işlemi şeklinde yorumlanır (Kafanızın karışmasına izin vermeyin fakat Pascal programlama dilinde bu operatör “:=” şeklini alacaktır)

Dolayısıyla “=” operatörünün sağındaki değer öncelikli olarak hesaplanır. Örneğin x değişkeninin bu ifadeye gelmeden önceki değeri “5” ise sağdaki değer hesaplandığında, sağ taraf “6” olur ve bu değer soldaki değişkene aktarılarak “x” değişkenin yeni değeri “6” yapılmış olur.

! “x=(x+1)” ifadesine gelmeden önce değeri “5” olan “x” değişkeni, bu ifadeden sonra yoluna değeri “6” olarak devam etmektedir.

2.6. Operatör Önceliği

İfadeler en az bir operatör içeren yapılardır. Bazı ifadelerde birden fazla operatör bulunabilmektedir.

? Peki birden fazla operatörün bulunduğu ifadelerde operatörlerin iş yapma sırası nasıl olacaktır?

Örneğin aşağıdaki ifadeyi inceleyelim,

```
c = 3+5*9
```

bu ifadedeki “c” değişkenine atanacak değer iki farklı şekilde olabilir. Birincisi c = (3+5)*9 = 72 ikincisi ise c = 3+(5*9) = 48. Bir ifadenin sadece bir değeri vardır. O halde bunların hangisi doğrudur? İşte tam bu noktada operatör önceliği kavramı önem kazanmaktadır. Daha önceden belirlenmiş birtakım mantıksal kurallar çerçevesinde operatörlere bir öncelik sırası verilmiştir. Hepimiz çok iyi biliyoruz ki çarpma operatörü her zaman toplama ve çıkarma operatöründen daha üstündür, yani önce çarpma işlemi yapılır. Bu yüzden örnekteki “c” değişkenine 3+(5*9) = 48 değeri atanır.

Günümüz programlama dillerinin hemen hemen hepsinde operatörlerin bir öncelik tablosu vardır. Böyle olmasaydı çok sayıda operatör içeren ifadelerin yorumlanması bir hayli zor olurdu.

İfadelerin parantez içine alınmış bölümleri her zaman önce çalıştırılırlar. Örneğin yukarıda verdiğimiz örneği aşağıdaki şekilde değiştirelim,

c = (3+5)*9

bu durumda ifade çalıştırıldığında c değişkenine 72 değeri atanacaktır. (Algan, 2008)

2.7. Algoritma Örnekleri

2.7.1. Örnek 1

Kullanıcı tarafından "şimdiki yıl" ve "doğum yılı" verilen kişinin yaşını bulan program.

1) Başla
2) Oku SimdikiYil
3) Oku DogumYili
4) YAS = SimdikiYil – DogumYili
5) Yaz YAS
6) Dur

2.7.2. Örnek 2

Kullanıcı tarafından "vize" ve "final" notları girilen bir öğrencinin ortalamasını hesaplayan program.

1) Başla
2) Oku Vize
3) Oku Final
4) ORT = (Vize*0.4) + (Final*0.6)
5) Yaz ORT
6) Dur

ÖZET

Bu ünitede programlama dillerinden bağımsız olarak, algoritma geliştirirken kullanılabilecek temel operatörlere değinildi.

Operatörler önceden tanımlanmış birtakım matematiksel ya da mantıksal işlemleri yapmak için kullanılan özel karakter ya da karakterler topluluğuydular. Operatörler tarafından işleme katılan değerlere ise operand denmekteydi.

Operatörleri işlevlerine göre gruplara ayırmıştık;

- Matematiksel Operatörler
- Karşılaştırma Operatörleri

- Mantıksal operatörler
- Atama Operatörü

Operatörlerde bir diğer önemli konu ise işlem önceliğiydi. Eğer ki operatörler arasında işlem önceliği olmasaydı (c = 3+5*9) gibi bir ifadenin iki farklı sonucu çıkabilmekteydi.

GÖZDEN GEÇİR

1. Operatörlerin olmadığı bir programlama düşünülebilir mi? Tartışınız.
2. Operatörlerin zenginliği (sayı ve çeşitlerinin çok oluşu) bir programlama diline neler katabilir?
3. Mantıksal operatörlerin, bilgisayar dünyasındaki yeri neresidir?

KAYNAKLAR

Algan, S. (2008). *Her Yönüyle C#.* İstanbul: Pusula Yayıncılık.