İSİM

bc – Keyfî duyarlılıkta hesaplama dili

KULLANIM

bc [**-hlwsqv**] [*uzun–seçenekler*] [*dosya* **...**]

SÜRÜM

Bu kullanım kılavuzu, GNU **bc** sürüm 1.06'ya göre hazırlanmıştır.

AÇIKLAMA

bc, girilen deyimleri işleterek keyfi uzunlukta ve duyarlılıktaki sayılar üzerinde matematiksel işlemler yapmaya imkan sağlayan etkileşimli bir ortam ve bir yazılım geliştirme dilidir. Sözdizimi açısından C yazılım geliştirme dilini andıran benzerlikler sözkonusudur. Komut satırından belirtmek suretiyle kullanabileceğimiz standart bir math kütüphanesine de sahiptir. Eğer gerekli ise; math kütüphanesi, belirtilen dosyalardan daha önce ele alınır. **bc** işleme tabi tutarken dosyaları komut satırında belirtilen sıraya göre ele alır. Belirtilen dosyaların tümü ele alınıp işlendikten sonra, standart girdiden girişler okunmaya başlar. Kodlar nasıl okunuyorlarsa bu halleriyle işletilirler. (Eğer belirtilen dosyalardan herhangi birinde **bc**'yi sonlandıracak bir komut yer alırsa, **bc** sonlanır, bir daha standart girdiden okuyamaz.)

bc'nin bu sürümü, geleneksel **bc** uygulamalarına ve POSIX taslak standardına göre ilave oluşumlar içermektedir. Komut satırından belirtilecek bir seçenekle bu genişletmeler kullanıma dahil edilebilir veya edilmeyebilir. Bu kılavuzda GNU **bc** anlatılacak ve ek oluşumlar yeri gelince belirtilecektir.

SEÇENEKLER

-h
--help
: Bir kullanım iletisi basar ve çıkar.

-i
--interactive
: Etkileşimli kipe geçilir.

-l
--mathlib
: Standart math kütüphanesini kullanıma sokar.

-w
--warn
: POSIX **bc**'ye ek oluşumlar için uyarılar verir.

-s
--standard
: Standart POSIX **bc** diline uygun olarak işlem yapar.

-q
--quiet
: Normalde gelen, GNU **bc** hoşgeldiniz iletisini görüntülemez.

-v
--version
: Sürüm ve telif hakkı bilgilerini gösterir ve çıkar.

SAYILAR

bc'deki en temel eleman sayılardır. Sayılar keyfi duyarlılıkta ifade edilirler. Yani **bc**'de bir sayı, tamsayı kısım ve ondalık kısım olarak ifade edilir. Tüm sayılar normalde iç hesaplamalarda onluk tabanda ele alınırlar ve tüm işlemler onluk taban üzerinden gerçekleştirilirler. (Bu sürüm, bölme ve çarpma işlemlerinde sayıların ondalık kısımlarını belli uzunluktan sonra kesmektedir.) **bc**'de sayıların iki özelliği vardır: uzunluk ve bölüntü. Uzunluk, sayıdaki tüm anlamlı rakamların, bölüntü ise ondalık noktadan sonraki rakamların adedidir.

Örneğin:

```
0.0000010 sayısının uzunluğu 7 ve bölüntüsü 6'dır.
01935.000 sayısının uzunluğu 7 ve bölüntüsü 3'tür.
```

Uzunluk açısından; en sağdaki sıfır anlamlı, en soldaki sıfır anlamsızdır.

DEĞİŞKENLER

Sayılar iki farklı türde değişken olarak saklanabilirler: basit değişkenler ve diziler. Her iki değişken türü için de değişkenlere birer ad verilir. Değişkenler bir harfle başlarlar ve sonrasında harfler, rakamlar ve altçizgi karakteri içerebilirler. Tüm harfler küçük harf olmak zorundadır. (Değişkenlere uzun isim verebilme bir genişletmedir. Çünkü, POSIX **bc**'de değişken isimleri tek bir küçük harftir.) Dizi değişkenleri ise isimden sonra [] (köşeli parantezler) alırlar.

bc'de dört tane özel değişken vardır. Bunlar; **scale**, **ibase**, **obase** ve **last**'dır. **scale**, işlemlerde sayılarda ondalık noktadan sonra kaç tane rakamın kullanılacağını belirtir. Öntanımlı **scale** değeri **0**'dır. **ibase** (input base) ve **obase** (output base) değişkenleri, sayılarda girdi ve çıktı için tabanları belirtir. Örneğin, ibase=10 ve obase=2 olursa sayılar onluk tabanda girilir ve sonuçlar ekrana ikilik tabanda yazdırılırlar. Öntanımlı **ibase** ve **obase** değerleri 10'dur. **last** değişkeni (bu bir genişletmedir) ise, en son ekrana yazdırılan sayıyı içeren bir değişkendir. Bu değişkenlere yeni değerler atanabilir. Bunlar ileride daha ayrıntılı olarak ele alınacaktır.

AÇIKLAMALAR

bc'de açıklama satırları **/*** karakterleri ile başlar ve ***/** karakterleri ile biter. Açıklamalar herhangi bir yerde başlayabilir ve girdide tek bir boşluk gibi ele alınır. (Yani, bu da açıklamaların girdi elemanları arasında bir ayraç olarak yorumlandığını gösterir. Dolayısıyla, bir değişken ismi içerisinde açıklama yer alamaz anlamındadır.) Açıklamalar satırsonu karakteri içerebilirler.

Ayrıca **bc**'de betiklerin kullanılmasını sağlamak için tek satırlık açıklamalar desteklenmektedir ve bu bir genişletmedir. Tek satırlık açıklama **#** karakterinden sonra başlar ve satır sonuna kadar devam eder. Satırsonu karakteri açıklamaya dahil değildir ve normal karakter olarak ele alınır.

İFADELER

Sayılar, ifadelerle ve deyimlerle birlikte kullanılırlar. Bu dil, etkileşimli bir çalışma ortamı olarak tasarlandığı için deyimler ve ifadeler de etkileşimli olarak işletilebilirler. **bc**'de bir 'main' işlevi kavramı yoktur. Bunun yerine kod, olduğu haliyle hemen yorumlanarak işletilir. (İşlevler, önce tanımlanırlar ve sonradan kullanılabilirler –ileride ele alınacaklar–).

En basit ifade bir 'sabit'tir. **bc**, **ibase** değişkeniyle belirtilen o anki girdi tabanını kullanarak sayıyı dahili onluk tabana çevirir. (Ancak, işlevler için bir istisna vardır.) **ibase** değişkeninin geçerli değerleri 2'den 16'ya kadardır. Bu aralığın dışında değerler atanırsa kendiliğinden 2 veya 16 olarak yeniden ayarlanır. Sayılar girilirken, 0–9 arası rakamlar ve A–F arası harfler ile kullanılabilir. (Not: A–F büyük harf olmalıdır. Çünkü, küçük harfler değişkenler için kullanılıyorlar.) Tek rakamlı sayılar, **ibase** ne olursa olsun, her zaman rakamın ifade ettiği değere sahip olurlar (örn, A=10)). **bc** çok rakamlı sayılarda, **ibase** değerine eşit veya daha büyük rakamları derhal **ibase-1** tabanında ele alarak sayının değerini hesaplar. Bu **FFF** sayısını daima, girdi tabanındaki en büyük 3 haneli sayı yapar.

İfadeler, diğer yüksek–düzeyli dillerdekilere oldukça benzerler. **bc**'de sayılar için farklı türler olmadığı için karışık türler için kurallar yoktur. Bunun yerine, sadece ifadelerin bölüntüleri üzerine kurallar vardır. Her ifadenin bir bölüntüsü vardır. Bu, orijinal sayıların bölüntülerinden, gerçekleştirilen işlemden ve çoğu durumda da **scale** değişkeninin değerinden elde edilir. **scale** değişkeninin alabileceği değerler, 0 (sıfır) ile C'deki tamsayı türü ile ifade edilebilen tamsayı aralığındadır.

Aşağıdaki geçerli ifadelerin açıklamalarında "ifade" komple bir ifadeyi ve "değişken" ise sıradan bir değişkeni veya bir dizi değişkenini göstermektedir. Bir basit değişken

isim

şeklinde ve bir dizi değişkeni ise aşağıdaki gibi belirtilir:

isim`[`*ifade*`]`

Özel olarak söz edilmediyse sonucun bölüntüsü, ilgili ifadelerin azami bölüntüsü olur.

−ifade
: Sonuç, *ifade*'nin negatifi olur.

++değişken
: *değişken*'in değerine önce 'bir' eklenir ve yeni değer ifadenin sonucu olur.

−−değişken
: *değişken*'in değeri önce 'bir' eksiltilir ve yeni değer ifadenin sonucu olur.

değişken++
: İfadenin sonucu değişkenin değerine göre hesaplanır, sonra *değişken*'in değeri 'bir' artırılır.

değişken−−
: İfadenin sonucu değişkenin değerine göre hesaplanır, sonra *değişken*'in değeri 'bir' eksiltilir.

ifade `+` *ifade*
: Sonuç, iki *ifade*'nin toplamıdır.

ifade `-` *ifade*
: Sonuç, iki *ifade*'nin farkıdır.

ifade `*` *ifade*
: Sonuç, iki *ifade*'nin çarpımıdır.

ifade `/` *ifade*
: Sonuç, iki *ifade*'nin bölümüdür. Sonucun bölüntüsü **scale** değişkeninin değeridir.

ifade `%` *ifade*
: Sonuç, "kalan"ı verir ve şu şekilde hesaplanır. `a%b`'yi hesaplarken, ilk önce a/b **scale** haneli olarak hesaplanır. **scale**`+scale(b)` ve `scale(a)`'dan hangisi daha büyükse bu bölüntüye göre `a-(a/b)*b` ifadesi sonucu hesaplamak için kullanılır. Eğer **scale** 0'a eşitlenirse ve her iki ifade de tamsayı ise, bu ifade tamsayı kalan işlevidir.

ifade `^` *ifade*
: İfadenin sonucu, birincinin ikinciye göre üssüdür. İkinci ifade bir tamsayı olmalıdır. (Eğer ikinci ifade tamsayı değilse, önce ekrana bir uyarı gelir ve ifade tamsayı olacak şekilde kesilir, yani ikinci ifadenin tam kısmı alınır). Sonucun bölüntüsü ise, eğer üs negatif ise **scale**'dir. Üs pozitif ise `scale(a^b) = min(scale(a)*b, max(scale, scale(a)))`'dır. Unutulmamalıdır ki *ifade*`^0` ifadesinin sonucu her zaman "1" olur.

`(` *ifade* `)`
: Parantezler, ifadenin değeri bulunurken standart önceliği değiştirir ve parantez içine alınan ifade daha önce hesaplanır.

değişken `=` *ifade*
: İfadenin sonucu değişkene atanır.

değişken <*işleç*>`=` *ifade*
: Bu, "*değişken* `=` *değişken* <*işleç*> *ifade*" ile eşdeğerdir ancak bir farkla; *değişken*'in değeri sadece bir kere elde edilir. Eğer *değişken* bir dizi ise, işlemin bir kere yapılacağı gözden uzak tutulmamalıdır.

İlişkisel ifadeler (karşılaştırma ifadeleri), sonuçları her zaman 0 veya 1 olan özel ifadelerdir. 0, yanlış (false) ve 1, doğru (true) olarak yorumlanır. Bunlar herhangi bir ifade içerisinde yer alabilirler. (POSIX **bc**'de ise ilişkisel ifadeler sadece **if**, **while** ve **for** deyimlerinde kullanılabilir ve sadece bir tane ilişkisel sınama olabilir.) İlişkisel işleçler şunlardır:

ifade1 `<` *ifade2*
: Sonuç, eğer *ifade1*, *ifade2*'den küçükse 1 olur. Aksi halde 0 olur.

ifade1 `<=` *ifade2*
: Sonuç, eğer *ifade1*, *ifade2*'den küçük ya da eşitse 1 olur. Aksi halde 0 olur.

ifade1 `>` *ifade2*
: Sonuç, eğer *ifade1*, *ifade2*'den büyükse 1 olur. Aksi halde 0 olur.

ifade1 `>=` *ifade2*
: Sonuç, eğer *ifade1*, *ifade2*'den büyük ya da eşitse 1 olur. Aksi halde 0 olur.

ifade1 `==` *ifade2*
: Sonuç, eğer *ifade1*, *ifade2*'ye eşitse 1 olur. Aksi halde 0 olur.

ifade1 `==` *ifade2*
: Sonuç, eğer *ifade1*, *ifade2*'den farklıysa 1 olur. Aksi halde 0 olur.

bc'de mantıksal (boolean) işlemler de geçerlidir. (POSIX **bc**'de mantıksal işlemler yoktur.) Mantıksal işlemlerin sonucu, ilişkisel işlemlerde olduğu gibi, 0 (false) yada 1 (true) olmaktadır. Mantıksal işleçler şunlardır:

`!`*ifade*
: *ifade* 0 ise sonuç 1'dir. Aksi halde 0 olur.

ifade `&&` *ifade*
: *ifade*'lerin ikiside sıfırdan farklıysa sonuç 1'dir. Aksi halde 0 olur.

ifade `||` *ifade*
: *ifade*'lerden biri sıfırdan farklıysa sonuç 1'dir. Aksi halde 0 olur.

İşleçlerin işlem öncelik sırası şöyledir (küçükten büyüğe):

```
|| işleci, önce sol taraf
&& işleci, önce sol taraf
! işleci, tarafsız
İlişkisel işleçler, önce sol taraf
Atama işleci, önce sağ taraf
```

```
+ and - işleci, önce sol taraf
*, / ve % işleci, önce sol taraf
^ işleci, önce sağ taraf
tek terimli - işleci
++ ve -- işleci
```

Bu önceliğin seçilmesinin nedeni, POSIX uyumlu **bc** yazılımlarının doğru çalışması içindir. Ancak, ilişkisel ve mantıksal işleçler, atama ifadelerinde kullanıldığında elverişsiz ve çok farklı bir durum ortaya çıkmaktadır. Aşağıdaki ifadeyi ele alalım:

```
a = 3 < 5
```

C programcıları bu ifadeyi ele alırken, önce `3<5` (sonuç 1'dir) ifadesini gerçekleştirir, ardından sonucu "a" değişkenine atarlar. **bc**'de ise önce "a"ya 3 atanır ve ardından 5 ile karşılaştırma yapılır. Buna dikkat etmek gerekir. En iyisi, ilişkisel ve mantıksal işleçler ile atama işleci aynı ifadede kullanıldığında karmaşayı önlemek için parantezleri kullanmaktır.

bc'de desteklenen biraz daha özel ifadeler vardır. Bunlar, standart işlevler ve kullanıcı tanımlı işlevlerde, "*isim* (*parametreler*) " şeklinde görülmektedirler. Ayrıntılı bilgi için *İŞLEVLER* (sayfa: 7) bölümündeki kullanıcı tanımlı işlevlere bakınız. Yerleşik işlevler şunlardır:

length (*ifade*)
: **length** işlevinin değeri, *ifade*'deki anlamlı rakamların adedidir.

read ( )
: **read** işlevi (bu bir genişletmedir), nerede kullanıldığına bakılmaksızın, standart girdiden bir sayı okumak için kullanılır. Ancak, standart girdiden veri ve yazılım birlikte okunurken problemlere neden olabileceğinden dikkatli olmak gerekir. Bu işlevin en iyi kullanım şekli, asla kullanıcıdan kod girmeyi gerektiren yerlerde değil, evvelce geliştirilen bir yazılım için kullanıcıdan bir girdi bekleyen yerlerde kullanmaktır. **read** işlevinin değeri, standart girdiden okunan sayının **ibase** tabanındaki değeridir.

scale (*ifade*)
: **scale** işlevinin değeri, *ifade*'denin sonucu olan sayıdaki ondalık hanelerin sayısıdır.

sqrt (*ifade*)
: **sqrt** işlevinin değeri, *ifade*'nin karekökü dür. *ifade*'nin sonucu negatif bir sayı ise bir çalışma anı hatası üretilir.

DEYİMLER

Deyimler, birçok cebirsel dilde olduğu gibi, ifadelerin sırayla değerlendirilmelerini sağlarlar. **bc**'de deyimler "mümkün olduğunca" işletilirler. İşletim, bir ya da daha fazla tam deyimden sonra bir satırsonu karakteri girildiğinde gerçekleşir. Bu bakımdan **bc**'de satırsonu karakterleri çok önemlidir. Aslında, deyimleri ayırmak için bir satırsonu karakteri ve bir noktalı virgül gerekir. Yanlış yerde kullanılmış bir satırsonu karakteri bir sözdizimi hatasına sebep olur. Satır sonu karakteri deyimler için bir ayraç olduğundan, bir satır sonu karakterini gizlemek için ters bölü karakteri kullanılabilir. Tersbölü karakterinden sonra gelen bir satırsonu karakterinden oluşan karakter çiftini **bc** boşluk karakteri olarak değerlendirir. Bir deyim listesi, noktalı virgül ve satırsonu karakteri ile sonlandırımış deyimlerden oluşur. Aşağıdaki listede **bc**'deki deyimler ve bunların ne yaptıkları anlatılmaktadır (Köşeli parantezler ([ ]) arasına alınanlar, deyimin isteğe bağlı kısımlarıdır.):

ifade
: Bu deyim iki şeyden birini yapar. Eğer ifade "<değişken> <atama> ..." şeklinde başlıyorsa, bunun bir atama deyimi olduğunu kabul eder. Eğer ifade bir atama deyimi değil ise, *ifade*'nin değeri bulunur ve çıktıya yazdırılır. Sayının ardından satırsonu karakteri yazdırılır. Örneğin, `a=1` ifadesi bariz bir atama

deyimidir; ve `(a=1)` içinde bir atama deyimi olan bir ifadedir. Çıktılanan tüm sayılar **obase** değişkeni ile belirtilen tabanda yazdırılırlar. **obase** değişkeninin alabileceği değerler 2 ile `BC_BASE_MAX` arasındadır. (Ayrıntılı bilgi için *SINIRLAR* (sayfa: 12) bölümüne bakınız.)

2 ile 16 arasındaki tabanlar için çıktı bildiğimiz yöntemle yazdırılır. 16'dan daha büyük tabanlar için ise, **bc** her bir haneyi yazdırmak için 10'luk tabanda birden çok karakter kullanır ve haneler arasında boşluk bırakır. Her hane, **obase−1** değerini 10'luk düzende yazmak için gereken sayı kadar rakamdan oluşur. Örneğin, `obase=20` için 65 sayısı ekrana "`03 05`" şeklinde yazdırılır. Eğer `obase=101` olsaydı, 25 sayısı ekrana "`025`" şeklinde yazdırılacaktı. Sayılar keyfi duyarlılıkta ele alındıkları için, bazı sayılar ekrana yazdırılırken tek bir satıra sığmayabilirler. Bu uzun sayılar yazdırılırken ekrana satırların sonuna "\" (tersbölü) karakteri gelir. Her bir satırda en fazla 70 karakter yazdırılabilir. **bc**'nin etkileşimli doğasından dolayı yazılan en son değer, özel **last** değişkeninde saklanır ki aynı sayıyı bir daha elde etmek için bir önceki *ifade*'yi tekrar yazmaya gerek kalmaz. **last** değişkenine atama yapılması da geçerlidir ve atanan değer son yazılan değerin üzerine yazılır. Yeni atanan değer, yeni bir değer yazılana kadar ya da **last** değişkenine yeni bir atama yapılana kadar geçerli kalır. (Bazı kurulumlar **last** için bir kısaltma olarak bir sayının parçası olmayan tek bir nokta (.) kullanımını mümkün kılabilir.)

dizge

dizge çıktıya yazdırılır. Çift tırnak "..." arasına alınan her şey (satırsonu karakteri de dahil) dizge olarak kabul edilir ve ekrana aynen yazdırılırlar. *dizge* yazdırıldıktan sonra satırsonu karakteri yazdırılmaz (yani aşağı satıra geçilmez, bu nedenle satırsonu karakteri *dizge*'nin içine yerleştirilmelidir).

print *liste*

print deyimi (bu bir genişletmedir) çıktıya birşeyler yazdırmak için kullanılan diğer bir yöntemdir. *liste* parametresi zorunludur, aralarına virgül (,) konularak dizgelerden ve ifadelerden oluşur. *liste*'deki her bir dizge veya ifade, yazıldıkları sıraya göre ekrana yazdırılır. Eğer belirtilmediyse satır sonunda satırsonu karakteri yazdırılmaz. İfadelerin değerleri bulunduktan sonra sonuçları ekrana yazdırılır. Bu esnada ekrana en son yazdırılan sayının değeri **last** değişkenine atanır. **print** deyimi içindeki dizgeler çıktıya yazdırılır ve özel anlamı olan karakterleri de içerebilirler. Özel karakterler tersbölü karakteri ile başlar. **bc** tarafından tanınan özel karakterler; "`a`" (uyarı; bip sesi), "`b`" (gerisilme), "`f`" (sayfa ileri), "`n`" (satırsonu), "`q`" (çift tırnak), "`t`" (sekme) ve "`\`" (tersbölü) karakterleridir. Bunların dışındaki karakterler gözardı edilir.

`{` *deyim_listesi*`}`

Bu, bir birleşik deyimdir. Bu yapı, birden çok deyimi gruplandırarak birlikte çalıştırmak için kullanılır. Deyimler arasında yukarıda anlatılan ayraçlar yer alırlar.

if `(` *ifade*`)` *deyim1*`[` **else** *deyim2* `]`

if deyimi *ifade*'nin değerini bulur ve i*fade*'nin sonucuna göre *deyim1*'i veya *deyim2*'yi çalıştırır. *ifade*'nin sonucu sıfırdan farklı ise *deyim1* işletilir. Eğer *deyim2* belirtilmişse ve *ifade*'nin sonucu da sıfır ise, *deyim2* işletilir (**else** sözcüğü bir genişletmedir).

while `(` *ifade*`)` *deyim*

while deyimi, *ifade*'nin değeri sıfırdan farklı olduğu müddetçe *deyim*'i tekrar tekrar işletir. Döngüde her yinelemede önce *ifade*'nin değeri bulunur, sonuç sıfırdan farklı ise *deyim* işletilir. *ifade*'nin sıfır olması durumunda veya çıkmak için **break** deyimi kullanıldığında döngü sonlanır.

for `(` `[`*ifade1*`]` `;` `[`*ifade2*`]` `;` `[`*ifade3*`]` `)` *deyim*

for deyimi, *deyim*'in tekrar tekrar çalıştırılmasını sağlar. İlk başta (yani döngü başlamadan önce) *ifade1* işletilir. Döngüde her yinelemede *deyim* işletilmeden önce *ifade2*'nin değeri bulunur. Eğer

sonuç sıfırdan farklı ise *deyim* yerine getirilir, eğer sonuç sıfır ise döngü sonlandırılır. Döngüde deyim yerine getirildikten sonra *ifade3* işletilir ve ardından *ifade2*'nin değeri tekrar bulunur. Ta ki *ifade2*'nin değeri sıfır oluncaya kadar... Döngüyü daha erken sonlandırmak için **break** deyimi kullanılabilir. Eğer *ifade1* veya *ifade3* kullanılmamışsa, bu noktada değerlendirilecek hiçbir şey olmadığından bir şey yapılmaz. Eğer *ifade2* kullanılmamışsa, *ifade2*'nin değeri 1 olarak kabul edilir. (POSIX **bc**'de her üç ifadeyi de belirtmek mecburidir. Bunların seçimlik kullanılması, bir genişletmedir.) **for** deyimine karşılık gelen deyimler şunlardır:

ifade1;
while (*ifade2*) {
  deyim;
  ifade3;
}

break
İçinde yer aldığı son **while** veya **for** döngüsünü sonlandırır.

continue
continue deyimi, (bir genişletmedir) içinde yer aldığı son **for** döngüsünü kendinden sonra gelen deyimler işletilmeden sonraki yinelemeden başlatır.

halt
halt deyimi, (bir genişletmedir) **bc**'yi sonladırır. **quit**'ten farkı, **halt** deyimi kesinkes değil, şart sağlandığında çalışır. Örneğin, "`if (0==1) halt`" deyimi **bc**'yi sonladırmaz; çünkü şart sağlanmadığı için **halt** deyimi işletilmez.

return
Bir işlevden 0 (sıfır) döndürerek çıkar. (Ayrıntılı bilgi için *İŞLEVLER* (sayfa: 7) bölümüne bakınız.)

return (*ifade*)
Bir işlevden *ifade*'nin değerini döndürerek çıkar. (Ayrıntılı bilgi için *İŞLEVLER* (sayfa: 7) bölümüne bakınız.) Bir genişletme olarak parantezler gerekli değildir.

YARDIMCI DEYİMLER

Yardımcı deyimler, bildiğimiz klasik anlamda işletilebilir deyimler değildirler. Bunlar "derleme" zamanında ele alınırlar.

limits
bc'nin yerel sürümü tarafından zorlanan yerel sınırları basar. Bu bir genişletmedir.

quit
Her nerede ve her ne şekilde kullanılırsa kullanılsın, **bc**'yi kesinkes sonlandırır. Örneğin, "`if (0==1) quit`" deyimi, şart sağlanmasa da **bc**'yi sonlandıracaktır.

warranty
Uzun bir garanti iletisi basar. Bu bir genişletmedir.

İŞLEVLER

İşlevler, daha sonra çalıştırılmak üzere hesaplamaların tanımlandığı bir yöntem sunar. **bc**'deki işlevler daima bir değer hesaplayıp bunu çağrıcıya döndürürler. İşlev tanımları "özdevimli"dir; yani, girdide saptanana kadar bir işlev tanımsızdır. Bu tanımlama, aynı isimde yeni bir işlev tanımlanıncaya kadar geçerlidir. Yeni işlev tanımı eskisinin yerine geçer. Bir işlev aşağıdaki gibi tanımlanır.

define *isim* (*parametreler*) { satırsonu
  auto_listesi *deyim_listesi* }

Bir işlev çağrısı ise "*isim* (*parametreler*) " biçiminde bir ifadedir.

Parametreler sayılar ya da dizilerdir (dizi parametreler bir genişletmedir). İşlev tanımında sıfır ya da daha fazla sayıda parametre verilebilir ve bunlar aralarına virgül konularak ayrılmış isimlerden oluşan bir liste olarak tanımlanır. Sayılar sadece değerleriyle çağrılan parametrelerdir. Diziler ise sadece değişkenleriyle çağrılırlar. Diziler, parametre tanımında "*isim* [ ] " şeklinde belirtilir. İşlev çağrısında asıl parametreler, sayı türünde ifadelerdir. Aynı yazım şekli, dizi parametrelerin tanımı olarak dizilerin aktarılmasında da kullanılır. Bir isimli dizi bir işleve değişkeni ile aktarılır. İşlev tanımları özdevimli olduklarından, parametre sayısı ve türleri işlev çağrıldığında kontrol edilir. Parametrelerin sayısı ve türlerindeki herhangi bir uyumsuzluk bir çalışma anı hatasına sebep olur. Ayrıca, tanımsız bir işlevin çağrılması da bir çalışma anı hatasına sebep olacaktır.

auto_listesi, "yerel" kullanım için isteğe bağlı bir değişken listesidir. *auto_listesi*'nin (eğer varsa) sözdizimi "**auto** *isim*, ... ; " şeklindedir (Sondaki noktalı virgül isteğe bağlıdır). Her *isim* bir **auto** değişkenin ismidir. Diziler, parametrelerde kullanılan sözdizimi ile belirtilebilir. Bu değişkenlerin sıfır ile ilklendirilerek, değerleri, işlevin başlangıcında yığıta basılır ve işlevin icrası boyunca kullanılır. İşlevden çıkışta bu değerler yığıttan çekilir, böylece değişkenler işlevin başlangıcındaki ilk değerlerine dönmüş olur. Parametreler gerçekte, işlev çağrısı sırasında ilklendirilen **auto** değişkenlerdir. **auto** değişkenler, geleneksel yerel değişkenlerden farklıdır. A işlevi, B işlevini çağırdığında, A işlevinin **auto** değişkenlerine, B işlevinde onlar **auto** değişkenler olarak çağrılmadıkça, B işlevi aynı isimlerle erişebilir. **auto** değişkenler ve parametreler yığıta basıldıklarından dolayı **bc** kendini çağrılan işlevleri destekler.

İşlevin gövdesi bir **bc** deyimleri listesidir. Tekrar belitelim ki, deyimler noktalı virgül ve satırsonu karakterleri ile birbirlerinden ayrılır. **return** deyimi işlevin sonlandırılmasına sebep olur ve bir değer döndürür. **return** deyiminin iki sürümü vardır. ilk şeklinde "**return**" ifadesi işlevi çağıran ifadeye sıfır değerini döndürür. İkinci şeklinde, "**return** (*ifade*)", *ifade*'nin değeri hesaplanıp çağıran ifadeye sonucu döndürülür. Her işlevin sonunda kendiliğinden bir **return(0)** vardır. Böylece bir işlevin sonunda açıkça bir **return** deyimi kullanmaya gerek kalmaksızın işlev sıfır değeri döndürerek sonlanır.

İşlevler ayrıca, **ibase** değişkeninin kullanımını da değiştirir. İşlev gövdesindeki tüm sabitler, işlev çağrısı sırasında **ibase** değişkeninin değeri kullanılarak dönüştürülür. Sayıların dönüşümü için daima **ibase**'in o anki değerini kullanan yerleşik **read** işlevi dışında, işlev icrası sırasında **ibase** değişiklikleri gözardı edilecektir.

Bir geliştirme olarak, tanım biçimi pek az esnektir. Standart, işlev gövdesini oluşturan ilk kuyruklu ayracın **define** ile aynı satırda olmasını ve kalan her şeyin alt satırlarda olmasını gerektirir. **bc**'nin bu sürümünde ilk kaşlı ayraçın öncesinde ya da sonrasında satırsonu karakteri kullanabilmeyi mümkün kılar. Örneğin aşağıdaki tanımlar geçerli tanımlardır:

```
define d (n) { return (2*n); }
define d (n)
  { return (2*n); }
```

MATH KÜTÜPHANESİ

bc, **-l** seçeneği ile çalıştırıldığında math kütüphanesini yükler ve öntanımlı bölüntü 20 yapılır. math işlvleri, çağrıldıkları sırada geçerli olan bölüntüye göre sonuç döndürürler. math kütüphanesi aşağıdaki işlevleri tanımlar:

s (*x*)
: Radyan cinsinden verilen *x*'in sinüsü.

c (*x*)
: Radyan cinsinden verilen *x*'in kosinüsü.

a (*x*)
: *x*'in arktanjantı; radyan cinsinden döner.

l (*x*)
: *x*'in tabii logaritması.

e (*x*)
: e üssü *x*.

j (*n*, *x*)
: *x* tamsayısına göre *n*. dereceden Bessel işlevi.

ÖRNEKLER

/bin/sh'da aşağıdaki atama, kabul değişkeni **pi**'ye pi değerini atar.

```
pi=$(echo "scale=10; 4*a(1)" | bc -l)
```

Aşağıdaki örnekte, math kütüphanesinde kullanılan üstel işlevin tanımı vardır. Bu işlev POSIX **bc**'de yazılmıştır.

```
scale = 20

/* e^x = (e^(x/2))^2 formülü kullanılıyor
  x yeterinde küçükse, bu seriyi kullanabiliriz:
    e^x = 1 + x + x^2/2! + x^3/3! + ...
*/

define e(x) {
  auto  a, d, e, f, i, m, v, z

  /* x'in işaretine bakalım. */
  if (x<0) {
    m = 1
    x = -x
  }

  /* x için önkoşul. */
  z = scale;
  scale = 4 + z + .44*x;
  while (x > 1) {
    f += 1;
    x /= 2;
  }

  /* Değişkenleri ilklendirelim. */
  v = 1+x
  a = x
  d = 1

  for (i=2; 1; i++) {
    e = (a *= x) / (d *= i)
    if (e == 0) {
```

```
      if (f>0) while (f--)  v = v*v;
      scale = z
      if (m) return (1/v);
      return (v/1);
    }
    v += e
  }
}
```

Aşağıdaki örnekte, çek defteri (checkbook) bakiyelerini hesaplayan basit bir yazılım verilmektedir. Yazılımı bir kez yazarak bir dosyaya kaydedebilirsiniz ve sonraları her seferinde yeniden yazmaksızın istediğiniz zaman kullanabilirsiniz.

```
scale=2
print "\nÇek defteri yazılımı!\n"
print "  Hatırlatma: Depozitolar negatif miktarlardır.\n"
print "  Çıkış için 0 yazın.\n\n"

print "Başlangıçtaki bakiye? "; bal = read()
bal /= 1
print "\n"
while (1) {
  "şu anki bakiye = "; bal
  "çekilecek miktar? "; trans = read()
  if (trans == 0) break;
  bal -= trans
  bal /= 1
}
quit
```

Aşağıdaki örnekte ise, kendi kendini çağıran bir faktöriyel hesaplama işlevi tanımlanmaktadır.

```
define f (x) {
  if (x <= 1) return (1);
  return (f(x-1) * x);
}
```

READLINE ve LIBEDIT SEÇENEKLERİ

GNU **bc** (bir yapılandırma seçeneği ile), GNU **readline** kütüphanesini ya da BSD **libedit** kütüphanesini kullanacak şekilde derlenebilir. Bu ise kullanıcıya, **bc**'ye göndermeden önce birden çok satır üzerinde düzenleme yapma imkanı sunar. Ayrıca, daha önceden girilen satırlar için bir geçmiş de tutar. Bu seçenek seçilirse, **bc** bir özel değişkene daha sahip olur. Bu özel **history** değişkeni, bellekte tutulacak satır sayısını tutar. –1 değeri sınırsız (bellek yettiği müddetçe) sayıda eski komut tutulacağını belirtir ve varsayılan değeri 100'dür. Eğer pozitif bir tamsayı belirtilirse, liste bu belirtilen sayı ile sınırlandırılmış olur. 0 değeri ise liste tutulmayacağını belirtir.

Daha ayrıntılı bilgi için, GNU **readline**, **history** kütüphanesi ile BSD **libedit** kütüphanesi hakkındaki belgeleri okuyunuz. **bc**, **readline** ve **libedit** kütüphaneleri aynı anda etkin olacak şekilde derlenemez.

FARKLILIKLAR

bc'nin bu sürümü, POSIX P1003.2/D11 taslağından uyarlanmıştır ve taslağa göre ve geleneksel **bc** uygulamalarına göre bir takım farklılıklar sözkonusudur. dc(1)kullanılan geleneksel yolla gerçekleştirilmemiştir. Bu sürüm, yazılımın kodlarını ayrıştırarak bayt koda çevirdikten sonra tek bir süreç olarak çalıştırır. Komut satırından verilen ve belgelendirilmemiş **-c** seçeneği ile, yazılım çalıştırılmaz, onun yerinee bayt kodları ekrana listelenir. Bu, aslında etkileşimli olarak ayrıştırıcının hatalarını görmek/gidermek ve math kütüphanesini hazırlamak için kullanılır.

POSIX **bc**'deki farklılıkların yanında bir de yeni özellikler eklenmiştir, bunlara da genişletmeler demekteyiz. Aşağıda, bahsedilen farklılıkların ve genişletmelerin bir listesi yer almaktadır:

`LANG`
: Bu sürüm `LANG` ve `LC_` ile başlayan ortam değişkenlerinin işlenmesinde POSIX standardına uyumlu değildir.

isimler
: Geleneksel ve POSIX **bc** değişkenler, diziler ve işlevler için sadece tek harften oluşan isimleri mümkün kılar. GNU **bc**'de bu isimler için çok karakterli isimler kullanılabilir. Bir isim bir harf ile başlar ve harfler, rakamlar ile altçizgi karakterini içerebilir.

dizgeler
: Dizgelerde boş karakter kullanılmasına izin verilmez, POSIX'e göre dizgelerde tüm karakterler kullanılabilmelidir.

last
: POSIX **bc**'de bir **last** değişkeni yoktur. **bc**'nin bazı gerçeklemelerinde bunun yerine nokta (.) kullanılır.

karşılaştırmalar
: POSIX **bc**'de karşılaştırmalar sadece **if**, **while** deyimlerinde ve **for** deyiminin ikinci ifadesinde kullanılabilir. Ayrıca bu deyimlerde sadece bir ilişkisel işleme izin verilir.

if deyimi ve **else** sözcüğü
: POSIX **bc**'de **else** sözcüğü yoktur.

for deyimi
: POSIX **bc**'de tüm ifadelerin deyim içinde kullanılması gereklidir.

`&&, ||, !`
: POSIX **bc**'de mantıksal işleçler yoktur.

read işlevi
: POSIX **bc**'de **read** işlevi yoktur.

print deyimi
: POSIX **bc**'de **print** deyimi yoktur.

continue deyimi
: POSIX **bc**'de **continue** deyimi yoktur.

return deyimi
: POSIX **bc**, **return** ifadesinin parantez içine alınmasını gerektirir.

dizi parametreler

POSIX **bc**'de (şimdilik) dizi parametreler tamamen desteklenmemektedir. POSIX sözdiziminde, işlev tanımlarında dizilere izin veriliyor, ancak bir dizinin bir işleve parametre olarak aktarılması desteklenmiyor. (Bu, aslında gözden kaçan bir dikkatsizliğin sonucudur.) Geleneksel **bc** gerçeklemeleri, dizileri sadece elemanlarının değerleri ile çağırabilmektedirler.

işlev biçimi
: POSIX **bc** ilk kaşlı ayracın **define** anahtar sözcüğünün bulunduğu satırda, **auto** anahtar sözcüğünün de alt satırında olmasını gerektirir.

`=+, =-, =*, =/, =%, =^`
: POSIX **bc** bu "eski tarz" atama işleçlerinin tanımlanmasını gerektirmez. Bu sürüm bu "eski tarz" atamalara izin verebilir. **limits** deyimini kullanarak kurulu sürümün bu atamaları desekleyip desteklemediğini öğrenebilirsiniz. Eğer destekliyorsa, "`a =- 1`" atamasında `a` değişkenine –1 atanmayacak, `a` değişkeninin değeri bir eksiltilecektir.

sayılardaki boşluklar
: **bc**'nin diğer gerçeklemeleri sayıların içinde boşluklara izin verir. "`x=1 3`" ataması, "`x=13`" olarak değerlendirilecektir. Aynı ifade bu sürümde bir sözdizimi hatasına sebep olacaktır.

çalıştırma ve hatalar
: Bu **bc**, yazılım içerinde sözdizimi ve diğer hataların bulunması durumunda kodun işletilmesi konusunda diğer gerçeklemelere göre oldukça farklılıklar içermektedir. Eğer bir işlev tanımlarken bir sözdizimi hatası yapılmışsa, hata düzeltme mekanizması deyimin başlangıcını bulmaya çalışır, ilk hatalı satırı bulur ve satır numarasını ekrana yazar; sonra yine de işlevin geri kalan kısımlarını ayrıştırmaya devam eder. İşlev içinde birkez bile hata yapılmışsa, o işlev 'çağrılamaz' addedilir ve tanımsız olur.

  Etkileşimli ortamda çalışırken bir sözdizimi hatası yapılırsa, bir uyarı gelir ve çalıştırılacak olan o anki blok geçersiz sayılır. İşletim bloğu, yazılışı tamamlanmış basit veya birleşik bir deyimden sonra satırsonu karakteri bulunan yapıdır. Örneğin,

```
a = 1
b = 2
```

  iki ayrı işletim bloğudur. Ancak,

```
{ a = 1
  b = 2 }
```

  tek işletim bloğudur. Oluşan herhangi bir çalışma anı hatası (error), o anki işletim bloğunu sonlandırır. Ancak, oluşan herhangi bir çalışma anı uyarısı (warning) işletim bloğunu sonlandırmaz.

kesmeler
: Bir etkileşimli oturumda, `SIGINT` sinyali (genelde klavyeden Ctrl–C'ye basıldığında üretilir) o anda işletilmekte olan bloğun yarıda kesilmesine neden olur. Hangi işlevin yarıda kesildiğini belirten bir çalışma anı hatası ekrana yazdırılır. Ardından, tüm çalışma anı yapılarının "temizlenme"sinin ardından, **bc**'nin yeni girdiler almak için hazır olduğunu yazan bir mesaj belirir. Önceden tanımlanmış olan tüm işlevler ve **auto**–olmayan değişkenler bu noktadan önceki değerleriyle aynen kalırlar. Tüm **auto** değişkenler ve işlev parametreleri bu "temizleme" işlemi esnasında silinirler. Etkileşimli olmayan bir oturumda ise `SIGINT` sinyali,**bc**'nin çalışmasını sonlandırarak kontrolü sisteme devredecektir.

SINIRLAR

Aşağıda **bc**'nin işlem yapabileceği sınırlar verilmiştir. Bunlardan bazıları aslında her bir kurulum için farklı farklı olabilir. Bunların geçerli değerlerini öğrenmek için **limits** deyimini kullanın.

`BC_BASE_MAX`
: Azami çıktı tabanı, 999 dur. Azami girdi tabanı ise 16 dır.

`BC_DIM_MAX`
: Dizilerde indisleme sınırı, keyfi olarak 65535 olarak belirlenmiştir. Kurulumunuzda farklı olabilir.

`BC_SCALE_MAX`
: Sayılarda ondalık noktadan sonraki hane sayısı `INT_MAX` ile sınırlanmıştır. Ayrıca ondalık noktadan önceki hane sayısı da `INT_MAX` ile sınırlanmıştır.

`BC_STRING_MAX`
: Dizgelerin içerebileceği karakter sayısı `INT_MAX` ile sınırlanmıştır.

üs
: Üstel değerlerde üssün sınırı `LONG_MAX`'tır.

değişken isimleri
: Basit değişken, dizi ve işlev isimlerinin her biri için eşsiz isim sayısı 32767 ile sınırlanmıştır.

ORTAM DEĞİŞKENLERİ

bc tarafından tanınan ortam değişkenleri şunlardır:

`POSIXLY_CORRECT`
: **-s** seçeneği ile aynıdır.

`BC_ENV_ARGS`
: Bu, **bc**'nin argümanları almak için kullandığı diğer bir mekanizmadır. Biçimi komut satırı argümanlarında olduğu gibidir. Bu argümanlar, belirtilen dosyalardan önce işlem görürler. Bu, kullanıcıya "standart" seçenekleri ayarlama imkanını sunar. Ortam değişkenlerinde belirtilen dosyalar tipik olarak içlerinde kullanıcının tanımladığı işlevler olan dosyalardır. Bu ayarlama ile artık, **bc** her çalıştırıldığında belirtilen dosyalar işleme sokulurlar.

`BC_LINE_LENGTH`
: Bu, sayılar ekrana yazdırılırken bir satırda kaç karakter olacağını belirten bir tamsayıdır. Uzun (yani, tek satıra sığmayan) sayılar için \ (tersbölü) ve satırsonu karakterleri bu miktara dahildir.

HATA AYIKLAMA

Eğer komut satırında belirtilen dosyalardan biri açılamazsa **bc** bunu raporlar ve hemen sonlanır. Ayrıca, derleme ve çalışma–anı hatalarında bunlar ekrana hata numaraları ile birlikte yazdırılırlar ve bu esnada açıklayıcı bilgiler verilir.

YAZILIM HATALARI

Hata bulma/düzeltme henüz çok iyi değil.

Hataları rapor etmek için konu alanına "bc" yazarak <bug–bc (at) gnu.org> adresine e–posta atınız.

YAZAN

Philip A. Nelson <philnelson (at) acm.org>

TEŞEKKÜRLER

Yazar kodun sınanmasındaki geniş yardımlarından dolayı Steve Sommars'a <Steve.Sommars (at) att.com> teşekkürlerinin kabulünü rica ediyor. Bir çok öneride bulundu ve onun katılımıyla bu çok daha iyi bir ürün oldu.

ÇEVİRENLER

Adem Güneş <adem (at) alaeddin.cc.selcuk.edu.tr>, Eylül 1999, v1.04
Nilgün Belma Bugüner <nilgun (at) belgeler·gen·tr>, Ocak 2004, v1.06

YASAL UYARI

Özgün belgedeki telif hakkı beyanı

us–mail: Philip A. Nelson
Computer Science Department, 9062
Western Washington University
Bellingham, WA 98226–9062

Bu dosya (man1-bc.pdf), belgenin XML biçiminin TEXLive ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

19 Ocak 2007