ALGOL 68

Geçmiş

Kökenleri

ALGOL 68, adından da anlaşılacağı gibi, 1960 yılında ilk kez resmileştirilmiş olan ALGOL dilinin bir devamı niteliğindedir. Piyasaya sürülmesinden kısa bir süre sonra, aynı yıl kurulan Uluslararası Bilgi İşlem Federasyonu (IFIP), ALGOL Çalışma Grubu'nu oluşturdu.

Uygulamalar

ALGOL 68-R

Standardın ilk uygulaması, 1968'in son rapor taslağına dayanarak, Birleşik Krallık'taki Kraliyet Radar Kuruluşu tarafından Temmuz 1970'te ALGOL 68-R olarak tanıtıldı. Ancak bu, tam dilin bir alt kümesi ve Barry Mailloux'du. .

Dilin ilk tam uygulaması 1974 yılında CDC Hollanda tarafından Control Data ana bilgisayar serisi için tanıtıldı. Bu, çoğunlukla Almanya ve Hollanda'da öğretmenlik yaparak sınırlı kullanım gördü.

68-R'ye benzer bir versiyon, Carnegie Mellon Üniversitesi'nden 1976'da 68S olarak tanıtıldı ve yine orijinalin çeşitli basitleştirmelerine dayanan ve DEC PDP-11 gibi daha küçük makinelerde kullanılması amaçlanan tek geçişli bir derleyiciydi. Aynı zamanda çoğunlukla öğretim amacıyla kullanılmıştır.

ALGOL 68 Zaman Çizelgesi

Yıl	Etkinlik	İştirakçi
Mar 1959	ALGOL Bülteni Sayı 1 (İlk)	Peter Naur / ACM
Şubat 1968	Taslak Rapor (DR) Yayınlandı	IFIP Çalışma Grubu 2.1
Mar 1968	Algol 68 Final Raporu r0 Münih Toplantısında Sundu	IFIP Çalışma Grubu 2.1
Haziran1968	İtalya'nın Tirrenia kentinde buluşma	IFIP Çalışma Grubu 2.1
Ağustos1968	Kuzey Berwick, İskoçya'da Toplantı	IFIP Çalışma Grubu 2.1
Aralık 1968	ALGOL 68 Final Raporu r0 Münih Toplantısında Sundu	IFIP Çalışma Grubu 2.1
Nisan 1970	ICL 1907F'de GEORGE 3 altında ALGOL 68-R (R)	Kraliyet İşaretleri ve Radar Est.
Eyl 1973	Algol 68 Revize Edilmiş Raporu r1 Yayınlandı	IFIP Çalışma Grubu 2.1
1975	ALGOL 68C (C) - taşınabilir derleyici (zcode VM)	Bourne,AndrewBirrell Michael Guy
Jun 1977	Strathclyde ALGOL 68 konferansı, İskoçya	ACM

Mayıs 1978	ALGOL H için Öneriler - ALGOL 68'nin Süper Dili	AP Siyah, VJ Rayward-Smith
1984	Sun, SPARC ve PC'ler için tam ALGOL 68S [(S)] derleyicisi	CH Lindsey ea, Manchester
Ağustos1988	ALGOL Bülteni Sayı 52 (son)	Ed. CH Lindsey / ACM
Mayıs 1997	İnternette yayınlanan Algol68 S [(S)]	Charles H. Lindsey
Kasım 2001	İnternette yayınlanan Algol 68 Genie (GNU GPL açık kaynaklı lisans)	Marcel van der Veer

Önemli dil elemanları

Birimler: İfadeler

Karmaşık tipler, çeşitli tip yapıcıları kullanarak daha basit olanlardan oluşturulabilir:

- ref *modu* - & C / C ++ ile benzer ve Pascal'da ref türünde bir *mod modu* referansı
- struct - C / C ++ 'daki struct ve Pascal'da record gibi yapılar oluşturmak için kullanılır
- union - C / C ++ ve Pascal'daki gibi birimler oluşturmak için kullanılır
- proc - C / C ++ fonksiyonlarını ve Pascal'daki prosedürleri / fonksiyonları tanımlamak için kullanılır.

Diğer bildirim sembolleri şunlardır: flex, heap, loc, ref, long, short, event

- flex - dizinin esnek olduğunu belirtir, yani talep üzerine uzayabilir.
- heap - değişkene genel öbekten bir miktar boş alan tahsis eder.
- loc - değişkene yerel yığının boş alanını ayır.
- long - bir int , real veya compl komutunun long boyutta olduğunu belirtir.
- short - bir int , real ya da compl komutunun short boyutta olduğunu belirtir.

Bir modun (tip) bir ismi, C / C ++ 'da typedef komutuna benzeyen ve Pascal dilinde yazılan bir mod bildirimi kullanılarak bildirilebilir:

```
int max=99;
mode newmode = [0:9][0:max]struct (
    long real a, b, c, short int i, j, k, ref real r
);
```

ALGOL 68 için sadece newmode mode göstergesinin eşittir sembolünün solunda göründüğünü ve en önemlisi yapımın önceliklere bakılmaksızın soldan sağa doğru görüntülendiğini unutmayın. Ayrıca, Algol 68 dizilerinin alt sınırının varsayılan olarak bir olduğunu, ancak - *max int* ile *max int* arasındaki herhangi bir tam sayı olabileceğini unutmayın.

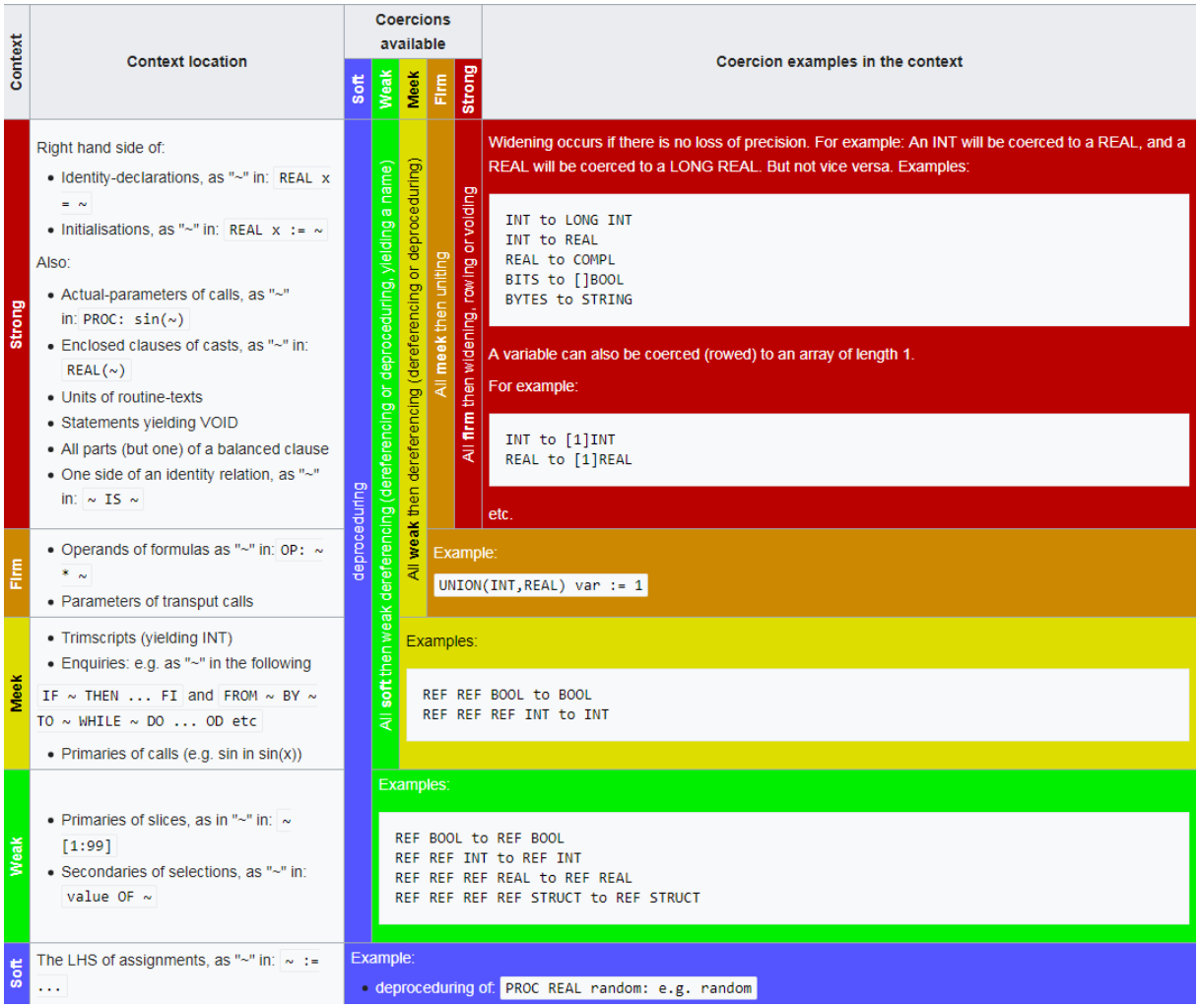
Mod bildirimleri türlerin özyinelemeli olmasına izin verir: doğrudan veya dolaylı olarak kendileri tarafından tanımlanır. Bu bazı kısıtlamalara tabidir - örneğin, bu beyanlar yasa dışıdır:

```
mode A = ref A
mode A = struct (A a, B b)
mode A = proc (A a) A
```

bunlar geçerliyken:

```
mode A = struct (ref A a, B b)
mode A = proc (ref A a) ref A
```

ALGOL 68, programda belirli bir noktada mevcut olan zorlama türlerini belirleyen bir bağlam hiyerarşisine sahiptir. Bu bağlamlar:



pr & co: Pragmats ve Yorumlar

Pragmatlar programdaki direktiflerdir , genellikle derleyiciye işaret eder; daha yeni dillerde bunlara "pragma" denir. Örneğin;

```
pragmat heap=32 pragmat
pr heap=32 pr
```

Yorumlar çeşitli şekillerde eklenebilir:

```
¢ The original way of adding your 2 cents worth to a program ¢
comment "bold" comment comment
co Style i comment co
# Style ii comment #
£ This is a hash/pound comment for a UK keyboard £
```

İfadeler ve bileşik ifadeler;

Kısa sembollerle seçim cümlesi örneği:

```
proc days in month = (int year, month)int:
  (month|
    31,
    (year÷x4=0 ∧ year÷x100≠0  ∨  year÷x400=0 | 29 | 28 ),
    31, 30, 31, 30, 31, 31, 30, 31, 30, 31
  );
```

Kalın sembollerle seçim cümlesi örneği:

```
proc days in month = (int year, month)int:
  case month in
    31,
    if year mod 4 eq 0 and year mod 100 ne 0  or  year mod 400 eq 0 then 29 else 28 fi,
    31, 30, 31, 30, 31, 31, 30, 31, 30, 31
  esac;
```

Seçim cümlesi örneği, *Kalın* ve *Kısa* sembollerin karışımı:

```
proc days in month = (int year, month)int:
  case month in
¢Jan¢ 31,
¢Feb¢ ( year mod 4 = 0 and year mod 100 ≠ 0  or  year mod 400 = 0 | 29 | 28 ),
¢Mar¢ 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ¢ to Dec. ¢
  esac;
```

ALGOL 68, herhangi bir sayıda ebattaki dizileri destekler ve bütün veya kısmi sıra veya sütunların dilimlenmesine izin verir.

```
mode vector = [1:3]    real;   # vector mode declaration (typedef)  #
mode matrix = [1:3,1:3]real;   # matrix mode declaration (typedef)  #
vector v1  := (1,2,3);         # array variable initially (1,2,3)   #
[]real v2   = (4,5,6);         # constant array, type equivalent to vector, bounds are implied  #
op + = (vector a,b) vector:    # binary operator definition         #
  (vector out; for i from ⌊a to ⌈a do out[i] := a[i]+b[i] od; out);
matrix m := (v1, v2, v1+v2);
print ((m[,2:]));              # a slice of the 2nd and 3rd columns #
```

Matrisler her iki şekilde de dilimlenebilir, örneğin:

```
  ref vector row = m[2,];  # define a ref (pointer) to the 2nd row #
  ref vector col = m[,2];  # define a ref (pointer) to the 2nd column #
```

Birlik halinde bir düğüm kullanım örneği:

Algol68[r0] as in the 1968 Final Report	Algol68[r1] as in the 1973 Revised Report

`node n := "1234";`
`real r; int i; compl c; string s`
`case r,i,c,s::=n in`
`  print(("real:", r)),`
`  print(("int:", i)),`
`  print(("compl:", c)),`
`  print(("string:", s))`
`  out print(("?:", n))`
`esac`
	`node n := "1234";`
	
	`case n in`
	`  (real r):   print(("real:", r)),`
	`  (int i):    print(("int:", i)),`
	`  (compl c):  print(("compl:", c)),`
	`  (string s): print(("string:", s))`
	`  out         print(("?:", n))`
	`esac`

İlişkili öncelikler ile Diyadik operatörler:

priority (Tertiary)	Algol68 "Worthy characters"[r0&r1]	+Algol68[r0&r1]	+Algol68[C,G]	+Algol68[r0]
9	+*, **i**	+×, ⊥		**!**
8	**shl**, **shr**, **, **up**, **down**, **lwb**, **upb**	↑, ↓, ⌊, ⌈		××, ^, **lws**, **ups**, ⌊, ⌈
7	*, /, %, **over**, %*, **mod**, **elem**	×, ÷, ÷×, ÷*, %×, ◻		÷:
6	-, +			
5	<, **lt**, <=, **le**, >=, **ge**, >, **gt**	≤, ≥		
4	**eq** =, **ne** ~= /=	≠, ¬=		
3	&, **and**	∧		/\
2	**or**	∨		\/
1	**minusab**, **plusab**, **timesab**, **divab**, **overab**, **modab**, **plusto**, -:=, +:=, *:=, /:=, %:=, %*:=, +=:	×:=, ÷:=, ÷×:=, ÷*:=, %×:=		**minus**, **plus**, **div**, **overb**, **modb**, ÷::=, **prus**

Transput: Input and output

Transput , ALGOL 68'in giriş ve çıkış tesislerine atıfta bulunmak için kullanılan terimdir. Biçimlendirilmemiş, biçimlendirilmiş ve ikili aktarım için önceden tanımlanmış prosedürleri içerir. Dosyalar ve diğer aktarma aygıtları tutarlı ve makineden bağımsız bir şekilde kullanılır. Aşağıdaki örnek, standard output aygıtına biçimlendirilmemiş bazı çıktılar yazdırır :

```
print ((newpage, "Title", newline, "Value of i is ",
  i, "and x[i] is ", x[i], newline))
```

Algol 68'in desteklediği paradigmalar Concurrent ve İmperative'dir. Eş zamanlı (concurrent) programlama, performanstan ziyade başka nedenlerden dolayı eş zamanlı işlemler arasında yeni iletişim tekniklerine dayanan bir yapıdadır. İşlemler arası iletişim genelde paylaşımlı hafıza veya mesaj geçirme tekniği ile yapılır.

Bellek yönetimi çöp toplamadır. Bellek yönetimi için geliştirilen çöp toplama aracı (Garbage Collection – GC), yeni nesnelere yer açmak (allocate) içi bellekteki uygun alanları serbest bırakan (deallocate) düzenektir. GC sistemi, uzun süre erişilmeyen, kullanılmayan nesneleri bellekten kaldırır, oluşturulan yeni nesneler için heap üzerinde yer açar.

Değişken kapsamı dinamik kapsamdır. Dinamik kapsam bağlama, bir ismin kapsamının, yordamların fiziksel olarak yakın olmalarına göre değil, yordamların çağrılma sırasına göre belirlenmesidir. Çalışma zamanında gerçekleşir.

Dinamik kapsam bağlamada bir ismin tanımı, çalışma sırasında aynı isimde bir başka tanımlama bulunana kadar kendisinden sonra çalıştırılan tüm komutlarda geçerlidir. Avantaj olarak, elverişliliği sağlar. Dezavantaj olarak okunabilirliği azaltır.

Tip sistemi; güçlü, tip kontrolü ; statik tip kontrolüdür.

Aritmetik işlem notasyonu ise infixtir. Halihazırda kullandığımız notasyondur. `3 + 4` örneğinde olduğu gibi, operatörler ifade içerisindedir. `3 + 4 x 5` örneğinde olduğu gibi, operatörler arasında işlem önceliği vardır. Verilen örnekte, ilk olarak çarpma işlemi, ardından toplama işlemi yapılır. İşlem sırasını değiştirmek için parantezler kullanılır. Örneğin, `(3 + 4) x 5`

Bazı problemlerin kaynak kodları:

Perfect number;

```
PROC is perfect = (INT candidate)BOOL: (
  INT sum :=1;
  FOR f1 FROM 2 TO ENTIER ( sqrt(candidate)*(1+2*small real) ) WHILE
    IF candidate MOD f1 = 0 THEN
      sum +:= f1;
      INT f2 = candidate OVER f1;
      IF f2 > f1 THEN
        sum +:= f2
      FI
    FI;
# WHILE # sum <= candidate DO
    SKIP
  OD;
  sum=candidate
);

test:(
  FOR i FROM 2 TO 33550336 DO
    IF is perfect(i) THEN print((i, new line)) FI
  OD
)
```

Output

```
           +6
        +28
       +496
      +8128
  +33550336
```

Binary search;

```
PROC iterative binary search = ([]ELEMENT hay stack, ELEMENT needle)INT: (
   INT out,
      low := LWB hay stack,
      high := UPB hay stack;
   WHILE low < high DO
      INT mid := (low+high) OVER 2;
      IF hay stack[mid] > needle THEN high := mid-1
      ELIF hay stack[mid] < needle THEN low := mid+1
      ELSE out:= mid; stop iteration FI
   OD;
      low EXIT
   stop iteration:
      out
```

Output

"A" near index 1

"Master" FOUND at index 4

"Monk" near index 8

"ZZZ" near index 8

Magic squares of odd order;

```
# construct a magic square of odd order                                  #
PROC magic square = ( INT order ) [,]INT:
   IF NOT ODD order OR order < 1
   THEN
      # can't make a magic square of the specified order                 #
      LOC [ 1 : 0, 1 : 0 ]INT
   ELSE
      # order is OK - construct the square using de la Loubère's        #
      # algorithm as in the wikipedia page                              #

      [ 1 : order, 1 : order ]INT square;
      FOR i TO order DO FOR j TO order DO square[ i, j ] := 0 OD OD;

      # as square [ 1, 1 ] if the top-left, moving "up" reduces the row  #
      # operator to advance "up" the square                              #
      OP   PREV = ( INT pos )INT: IF pos = 1 THEN order ELSE pos - 1 FI;
      # operator to advance "across right" or "down" the square          #
      OP   NEXT = ( INT pos )INT: ( pos MOD order ) + 1;
```

```
        # fill in the square, starting from the middle of the top row       #
        INT col := ( order + 1 ) OVER 2;
        INT row := 1;
        FOR i TO order * order DO
            square[ row, col ] := i;
            IF square[ PREV row, NEXT col ] /= 0
            THEN
                # the up/right position is already taken, move down         #
                row := NEXT row
            ELSE
                # can move up and right                                     #
                row := PREV row;
                col := NEXT col
            FI
        OD;

        square
    FI # magic square # ;

# prints the magic square                                                   #
PROC print square = ( [,]INT square )VOID:
    BEGIN
        INT order = 1 UPB square;
        # calculate print width: negative so a leading "+" is not printed   #
        INT width := -1;
        INT mag   := order * order;
        WHILE mag >= 10 DO mag OVERAB 10; width MINUSAB 1 OD;
        # calculate the "magic sum"                                         #
        INT sum := 0;
        FOR i TO order DO sum +:= square[ 1, i ] OD;
        # print the square                                                  #
        print( ( "maqic square of order ", whole( order, 0 ), ": sum: ", whole( sum, 0 ), newline ) );
        FOR i TO order DO
            FOR j TO order DO write( ( " ", whole( square[ i, j ], width ) ) ) OD;
            write( ( newline ) )
        OD
    END # print square # ;

# test the magic square generation                                          #
FOR order BY 2 TO 7 DO print square( magic square( order ) ) OD
```

Output

```
maqic square of order 1: sum: 1
 1
maqic square of order 3: sum: 15
 8 1 6
 3 5 7
```

```
 4 9 2
maqic square of order 5: sum: 65
 17 24  1  8 15
 23  5  7 14 16
  4  6 13 20 22
 10 12 19 21  3
 11 18 25  2  9
```

Caesar cipher;

```
MODE MODXXVI = SHORT SHORT INT; # MOD26 #

  PROC to m26 = (CHAR c, offset)MODXXVI:
  BEGIN
    ABS c - ABS offset
  END #to m26#;

  PROC to char = (MODXXVI value, CHAR offset)CHAR:
  BEGIN
    REPR ( ABS offset + value MOD 26 )
  END #to char#;

  PROC encrypt = (STRING plain, MODXXVI key)STRING:
  BEGIN
    [UPB plain]CHAR ciph;
    FOR i TO UPB plain DO
      CHAR c = plain[i];
      ciph[i]:=
       IF "A" <= c AND c <= "Z" THEN
          to char(to m26(c, "A")+key, "A")
       ELIF "a" <= c AND c <= "z" THEN
          to char(to m26(c, "a")+key, "a")
       ELSE
          c
       FI
    OD;
    ciph
  END #encrypt#;

# caesar main program #
  STRING text := "The five boxing wizards jump quickly" # OR read string #;
  MODXXVI key := 3; # Default key from "Bello Gallico" #

  printf(($gl$, "Plaintext ---------->" + text));
```

```
  text := encrypt(text, key);
  printf(($gl$, "Ciphertext ---------->" + text));
  printf(($gl$, "Decrypted Ciphertext ->" + encrypt(text, -key)))

END #caesar#
```

Output

```
Plaintext ---------->The five boxing wizards jump quickly

Ciphertext ---------->Wkh ilyh eralqj zlcdugv mxps txlfnob

Decrypted Ciphertext ->The five boxing wizards jump quickly
```

Kaynakça:

https://kodcu.com/2011/07/statik-tipli-diller-ile-dinamik-tipli-diller-arasindaki-farklar/

http://www.wikizero.biz/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvQUxHT0xfNjg

https://www.selamigungor.com/post/20/infix-prefix-postfix-notasyonu

https://rosettacode.org/wiki/Rosetta_Code