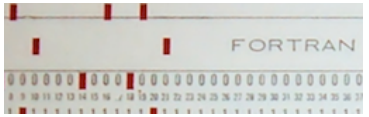
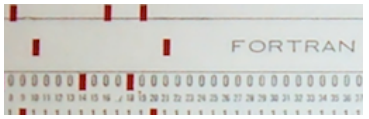


Programowanie współbieżne... (3)

Andrzej Baran **2010/11**

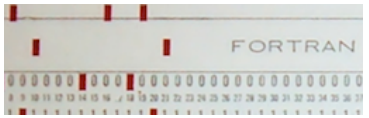
LINK: http://kft.umcs.lublin.pl/baran/prir/index.html

Fortran – minimum



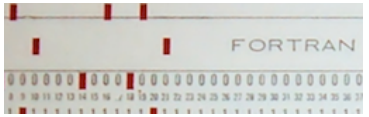
- M. Metcalf: Fortran 90/95/HPF Information File
  http://www.fortran.com/metcalf.htm
- M. Metcalf, Fortran 90 Tutorial
  http://wwwasdoc.web.cern.ch/wwwasdoc/WWW/f90/f90.html
- **M. Metcalf's Fortran 90 CNL Articles**
  http://wwwasdoc.web.cern.ch/wwwasdoc/f90.html
- Fortran Tutorials: http://www.fortran.com/
- Fortran 90/95 Explained, M. Metcalf and J. Reid, (Oxford, 1996)
- Fortran 95/2003 Explained), M. Metcalf, J. Reid, M. Cohen (Oxford University Press, 2004)

Fortran...



- 1954 FORmula TRANslator, IBM
- 1961 FORTRAN IV – przenośna wersja
- 1966 FORTRAN 66 – pierwszy standard
- 1977 FORTRAN 77 – gross oprogramowania
- 1990 Fortran 90 – rozszerzenie F 77 (dynamiczna alokacja tablic, free form source)
- 1995 Fortran 95 – małe poprawki do standardu
- 2003 Fortran 2003 – rozszerzenie (obiekty; duże podobieństwo z C; brak kompilatorów)

Arkusz do kodowania



IBM

FORTRAN Coding Form

X28-7327-5
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

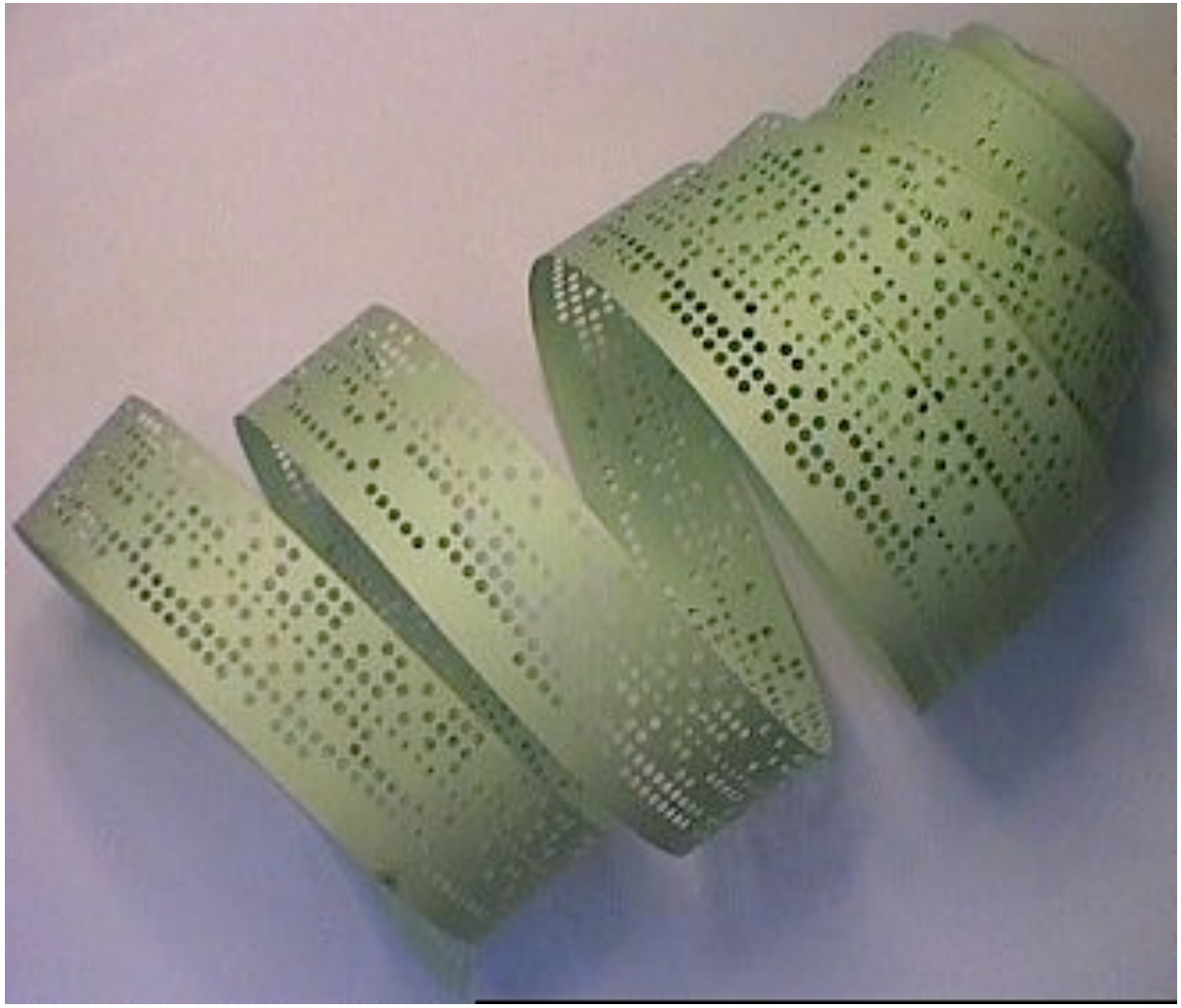
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER | CONT. | FORTRAN STATEMENT | IDENTIFICATION SEQUENCE

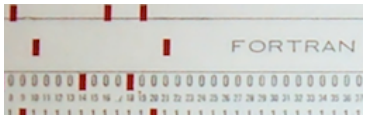
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

* A standard card form, IBM electro 888157, is available for punching statements from this form

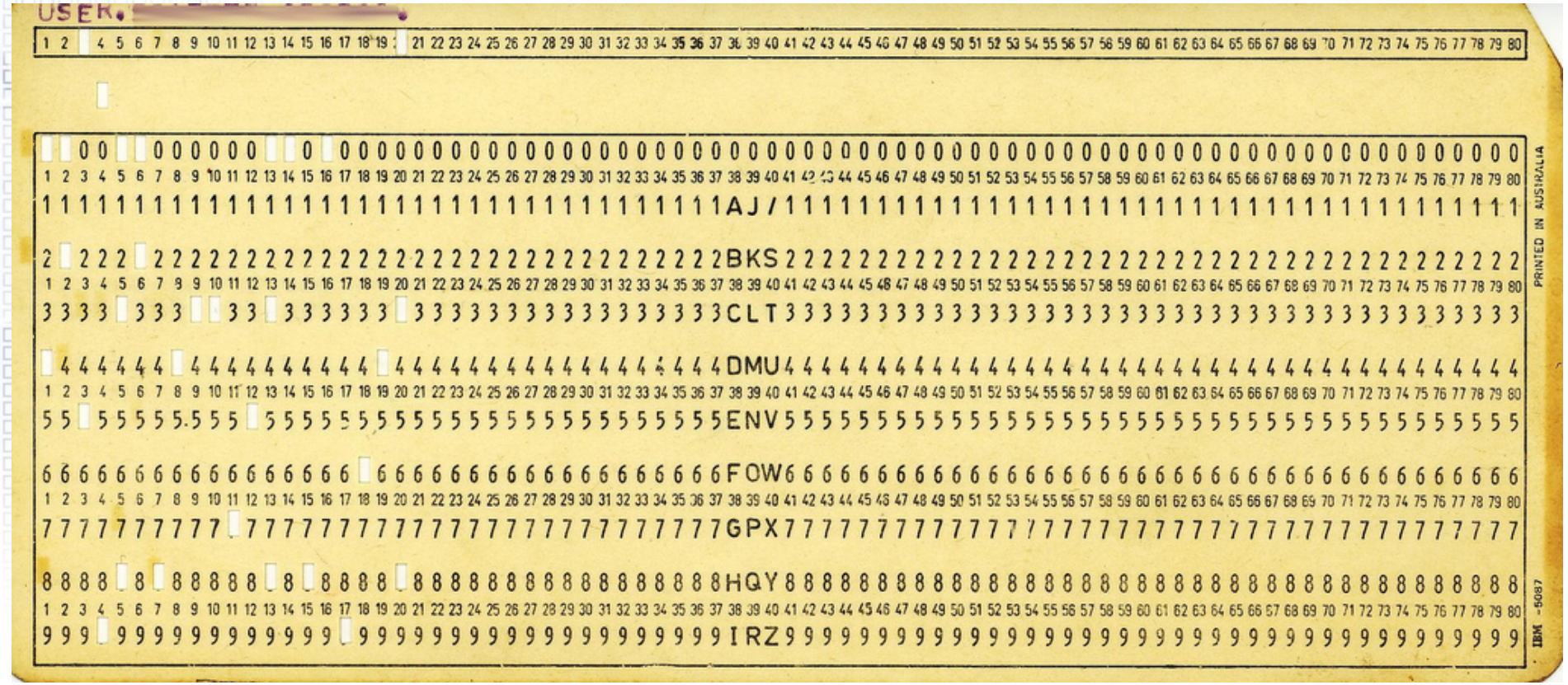
Kodowanie



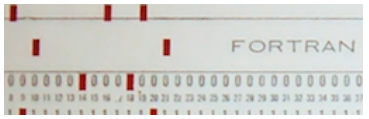
Instrukcje - wprowadzanie...



Jedna instrukcja Fortranu = karta (~8cm x 19cm) (CYBER, lata >1970)



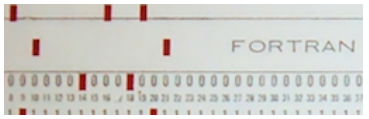
Instrukcje - wprowadzanie...



Lepsze czasy – napisy na karcie do kodowania (CDC)

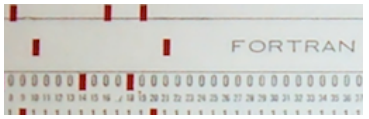
```
      DO I=1,N
      ▮  ▮
       ▮     ▮
000000000000▮0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
111111111111▮1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
333333333333▮3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
444444▮4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
555555555555▮5555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
6666666▮66▮6666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666
777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777
888888888▮8▮8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
999999999▮9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
```

Formaty



- Format ustalony (rygorystyczny)
- Format swobodny (free) – bardziej czytelny

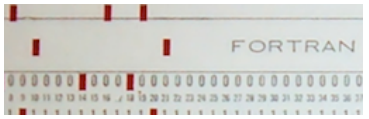
Formaty




```
      program hello_world
C       Format ustalony programu w Fortranie
        implicit none
        print *,
       &                "Hello World"
C       Ostatnia linia jest kontynuacją poprzedniej
     end program hello_world
```

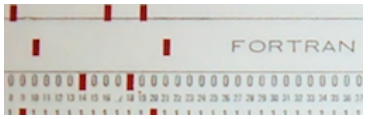
```
program hello_world
  ! Format swobodny programu w Fortranie
  implicit none
  print *, &
        "Hello World"
  ! Ostatnia linia jest kontynuacją poprzedniej
end program hello_world
```

Przykład




```
C PRZYKLAD PROGRAMU – POLE TROJKATA
C
WRITE(6,10)
10    FORMAT(/,' WPROWADZ TRZY LICZBY')
READ(5,25) A, B, C
25  FORMAT(3F4.2)
S=(A+B+C)/2.0
POLE=SQRT(S*(S-A)*(S-B)*(S-C))
WRITE(6,17) POLE
17   FORMAT(' POLE TROJKATA=',F8.3)
STOP
END
```

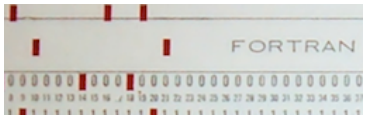
Fortran90



metcalf/WWW/f90

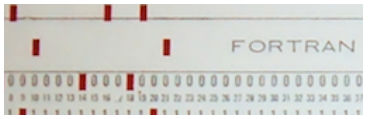
(wykład; jeśli brak dostępu do sieci)

Przykład



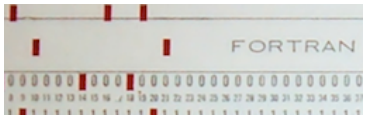

```
FUNCTION string_concat(s1, s2)     ! This is a comment
  TYPE (string), INTENT(IN) :: s1, s2
  TYPE (string) string_concat
  string_concat%string_data = s1%string_data(1:s1%length) // &
     s2%string_data(1:s2%length)  ! This is a continuation
  string_concat%length = s1%length + s2%length
END FUNCTION string_concat
```

Typy




```
INTEGER, PARAMETER :: two_bytes = SELECTED_INT_KIND(4)
INTEGER, PARAMETER :: long = SELECTED_REAL_KIND(9, 99)

INTEGER(KIND=2) i
REAL(KIND=long) a
COMPLEX current
LOGICAL Pravda
CHARACTER(LEN=20) word
CHARACTER(LEN=2, KIND=Kanji) kanji_word
```

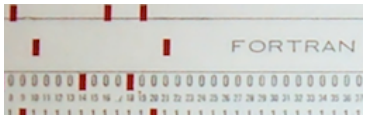
Typy złożone




```
TYPE person
  CHARACTER(10) name
  REAL       age
END TYPE person

TYPE(person) you, me
you%age
you = person('Smith', 23.5)

TYPE point
  REAL x, y
END TYPE point
TYPE triangle
  TYPE(point) a, b, c
END TYPE triangle

TYPE(triangle) t
```

Elementy typu TRIANGLE: t%a t%b t%c
wierzchołki (skalary): t%a%x t%a%y t%b%x etc.

Typy złożone



TABLICE

```
REAL a(10)
INTEGER, DIMENSION(0:100, -50:50) :: map ! Map jest tablicą
```

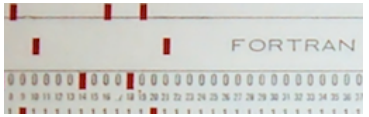
Elementy skalarne tablicy a: a(1) a(i*j)

```
TYPE triplet
  REAL, DIMENSION(3) :: vertex !Vertex jest tablicą o 3 elementach
END TYPE triplet

TYPE(triplet), DIMENSION(4) :: t
```

t(2) jest skalarem, t(2)%vertex jest tablicą skalarów

- `CHARACTER(80), DIMENSION(60) :: page ... = page(j)(i:i)`
  ! napisy są dozwolone; mamy podnapis '0123456789'(i:i) you%name(1:2)
- Dozwolone są napisy o zerowej długości: page(j)(i:i-1) ! zero-length string

Instrukcje kontrolne




```
SELECT CASE (number) ! NUMBER of type integer     ! Instrukcja CASE
   CASE (:-1) ! all values below 0
      n_sign = -1
   CASE (0) ! only 0
      n_sign = 0
   CASE (1:) ! all values above 0
   n_sign = 1
END SELECT
```

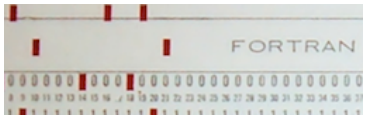
PĘTLE DO

```
outer:  DO
inner:        DO i = j, k, l ! only integers
                 :
                 IF (...) CYCLE
                 :
                 IF (...) EXIT outer
              END DO inner
        END DO outer
```

WYRAŻENIA SKRÓCONE

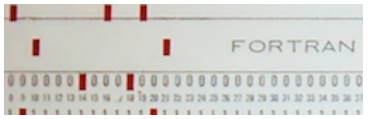
```
tot = SUM( a(m:n) )        ! Suma elementów tablicy a o indeksach od m do n
```

Funkcje - przykład




```
function norm(vector)
   ! 2-norma of wektora
   real :: norm
   real, intent(in), dimension(:) :: vector
   integer :: i
   norm=0 do i=1,size(vector)
   norm=norm+vector (i)**2 end do
   norm=sqrt(norm)
end function norm
```

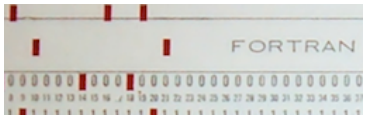
Moduły



Modułów używamy do przechowywania

- danych (zastępują COMMON oraz BLOCK DATA);
- definicji typów
- podprogramów (zastąpienie m. in. ENTRY);
- bloków interfejsów;
- grup namelist

Moduły



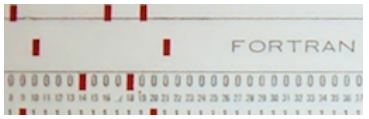
Przykład modułu zawierającego definicję typu, blok interfejsu oraz funkcję (podprogram):

```
MODULE interval_arithmetic
    TYPE interval
        REAL lower, upper
    END TYPE interval
    INTERFACE OPERATOR(+)
        MODULE PROCEDURE add_intervals
    END INTERFACE
        :
    CONTAINS
    FUNCTION add_intervals(a,b)
        TYPE(interval), INTENT(IN) :: a, b
        TYPE(interval) add_intervals
        add_intervals%lower = a%lower + b%lower
        add_intervals%upper = a%upper + b%upper
    END FUNCTION add_intervals      ! FUNCTION mandatory
    :
END MODULE interval_arithmetic
```

Sposób na dostęp do modułu (w programie lub podprogramie, pierwsza deklaracja):

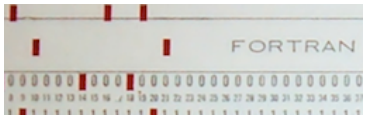
```
USE interval_arithmetic
```

Tablice – alokacja - dealokacja




```
MODULE work_array
  INTEGER n
  REAL, DIMENSION(:,:,:), ALLOCATABLE :: work
END MODULE
PROGRAM main
  USE work_array
  READ (*, *) n
  ALLOCATE(work(n, 2*n, 3*n), STAT=status)
  :
  DEALLOCATE (work)
```

Tablice



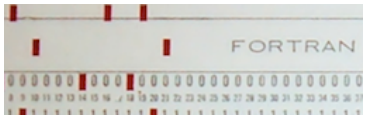
- Operacje podstawowe

```
REAL, DIMENSION(10) :: a, b
a = 0.            ! scalar broadcast; elemental assignment
b = sqrt(a)       ! intrinsic function result as array object
```

- Wynik działania funkcji = tablica

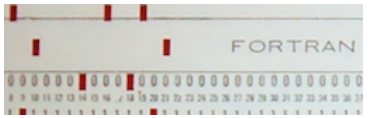
```
PROGRAM test
   REAL, DIMENSION(3) :: a = (/ 1., 2., 3./),  b = (/ 2., 2., 2. /),  r
   r = f(a, b)
   PRINT *, r
CONTAINS
   FUNCTION f(c, d)
   REAL, DIMENSION(:) :: c, d
   REAL, DIMENSION(SIZE(c)) :: f
   f = c*d          ! (or some more useful function of c and d)
   END FUNCTION f
END PROGRAM test
```

Tablice - operacje



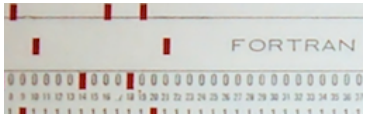

```
real :: a(3,2),b(3,2),c(2,2),x(3),y(3),z(2),e
x=(/1 ,2 ,3/)  !Wektorowy literał stały(/    /).

a=reshape((/1.5,.7,.3,.4,.6,.8/),shape(a))

c=matmul(transpose(a), b) ! Macierz a' * b.

z=matmul(x,a) ! Wektor x macierz: x' * b
e=matmul(a,(/1.2 ,1.5/)) ! Macierz a razy wektor.
e=dot_product(z,b(2 ,:))

print '(2f15.8)', transpose(a) ! Format i wydruk a'
e=size(a)   ! e jest równe 6, wymiar(a)
z = shpe(a) ! z is (/3 ,2/).
```

Tablice - operacje




```
logical ,dimension (10 ,5):: l

real dimension (10,5) a

print *, any(a==0) ! Jeśli element a jest 0.

print *, all(a==0) ! Jeśli wszystkie el. a są 0.

print *, count(l) ! Liczba elementów l.

print *, maxval(a,2) ! Wektor maksymalnych wartości z wierszy a.

print *, minval(a) ! Minimalny element a

print *, product(a(1:3,:)) ! Iloczyn 3 wierszy a.

print *, sum(a,1) ! Wektor sum kolumn a.
```

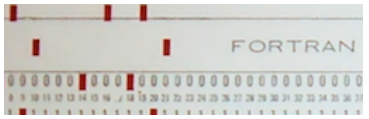
„where” i „forall”




```
where (a<0.0)
   a=0.0 ! Zeruj all wszystkie ujemne elementy a
end where
```

```
forall   ( i =1:20)
   a(i)= 0.5*i
end forall
```

Tablice, instr. wewnętrzne



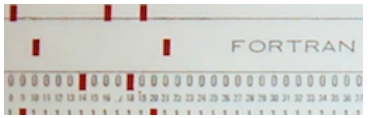
Vector and matrix multiply

DOT_PRODUCT	Dot product of 2 rank-one arrays
MATMUL	Matrix multiplication

Array reduction

ALL	True if all values are true
ANY	True if any value is true. Example: IF (ANY(a > b)) THEN
COUNT	Number of true elements in array
MAXVAL	Maximum value in an array
MINVAL	Minimum value in an array
PRODUCT	Product of array elements
SUM	Sum of array elements

Tablice, instr. wewnętrzne



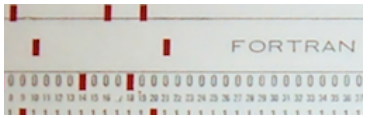
Array inquiry

ALLOCATED	Array allocation status
LBOUND	Lower dimension bounds of an array
SHAPE	Shape of an array (or scalar)
SIZE	Total number of elements in an array
UBOUND	Upper dimension bounds of an array

Array construction

MERGE	Merge under mask
PACK	Pack an array into an array of rank
SPREAD	Replicate array by adding a dimension
UNPACK	Unpack an array of rank one into an array under mask

Tablice, instr. wewnętrzne



Array reshape

RESHAPE	Reshape an array

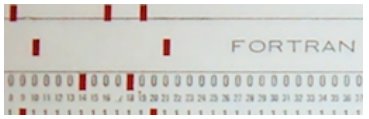
Array manipulation

CSHIFT	Circular shift
EOSHIFT	End-off shift
TRANSPOSE	Transpose of an array of rank two

Array location

MAXLOC	Location of first maximum value in an array
MINLOC	Location of first minimum value in an array

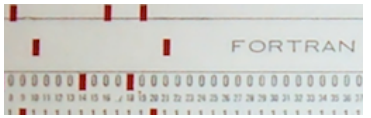
Precyzja obliczeń



WAŻNE informacje o dokładności obliczeń (funkcje)

DIGITS(X)	Number of significant digits
EPSILON(X)	Almost negligible compared to one (real)
HUGE(X)	Largest number
MAXEXPONENT(X)	Maximum model exponent (real)
MINEXPONENT(X)	Minimum model exponent (real)
PRECISION(X)	Decimal precision (real, complex)
RADIX(X)	Base of the model
RANGE(X)	Decimal exponent range
TINY(X)	Smallest positive number (real)

EPSILON, TINY

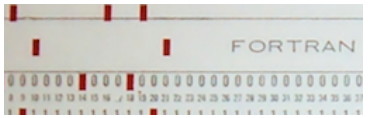



```
Program epstiny
  x=1.e0
  eps = epsilon(x)
  tin = tiny(x)
  write(*,*) eps, tin
  write(*,*) eps+1, tin+1                  ! Standard format
  write(*,"(2e20.12)") eps+1, tin+1        ! More exact format
end program epstiny
```

Wynik działania programu:

```
 1.1920929E-07  1.1754944E-38
  1.000000       1.000000
 0.100000011921E+01  0.100000000000E+01
```

Procedury wewnętrzne



- Bit inquiry

`BIT_SIZE`	`Number of bits in the model`

- Bit manipulation

`BTEST`	`Bit testing`
`IAND`	`Logical AND`
`IBCLR`	`Clear bit`
`IBITS`	`Bit extraction`
`IBSET`	`Set bit`
`IEOR`	`Exclusive OR`
`IOR`	`Inclusive OR`
`ISHFT`	`Logical shift`
`ISHFTC`	`Circular shift`
`NOT`	`Logical complement`

Procedury wewnętrzne

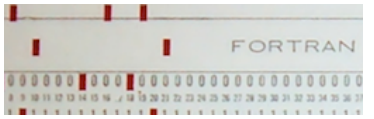
- Transfer function, as in

```
INTEGER :: i = TRANSFER('abcd', 0) ! replaces part of EQUIVALENCE
```

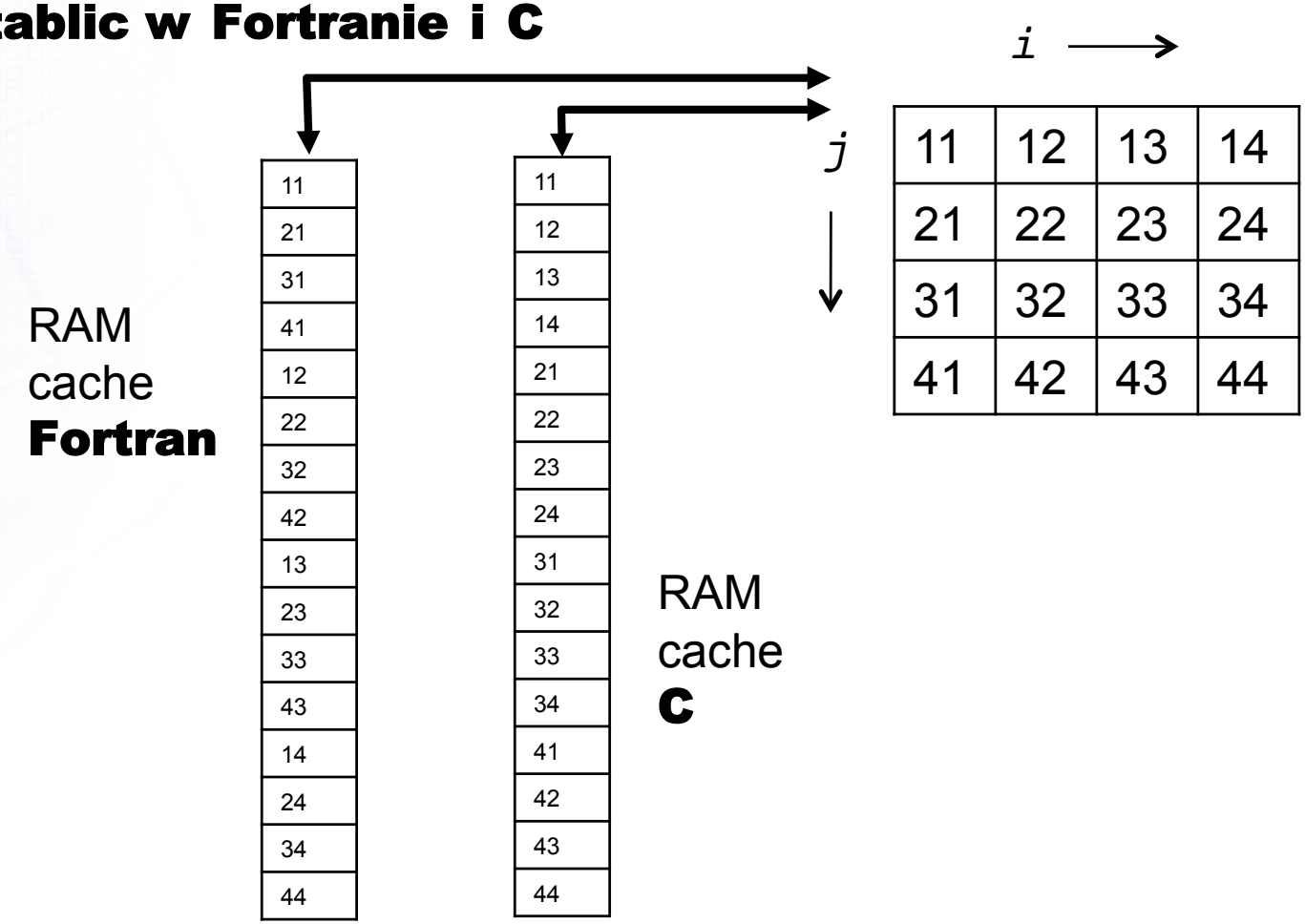
- Subroutines

```
DATE_AND_TIME      Obtain date and/or time
MVBITS             Copies bits
RANDOM_NUMBER      Returns pseudorandom numbers
RANDOM_SEED        Access to seed
SYSTEM_CLOCK       Access to system clock
```

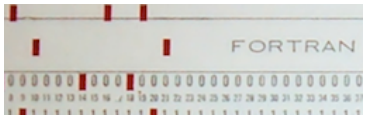
Tablice – szybkość obliczeń



Struktura tablic w Fortranie i C



Tablice – szybkość obliczeń



Konsekwencje. Organizacja pętli „do” lub „for”

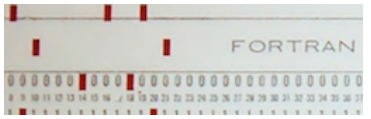
```
do i=1,n
   do j=1,n
      ...A(i,j)...
   end do
end do
```

Fortran : gorzej
C : LEPIEJ

```
do j=1,n
   do i=1,n
      ...A(i,j)...
   end do
end do
```

Fortran : LEPIEJ
C : gorzej

Zadania (Lab)

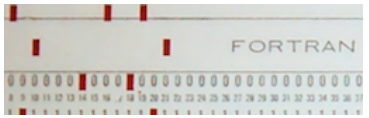


- Porównaj czasy wykonania różnych typów pętli ijk, ikj, kji,itd, na przykład pętla ijk wygląda tak:

```
 do i=1,n
   do j=1,n
     do k=1,n
        coś_oblicz_z_elem_tablicy a(i,j,k)
     end do
     ...
```

Przyjmij raczej dużą wartość n. Wnioski. Powtórz testy wielokrotnie (Średnie są ważne, a nie pojedynczy wynik. Procesor w czasie obliczeń może być zajęty czymś innym...)

Szybkość obliczeń...



W konsekwencji:

Szybkość obliczeń zależy od cech charakterystycznych danego języka programowania, a nie tylko od szybkości procesorów oraz urównoleglania źle napisanych programów sekwencyjnych☺

Problemy...?