Historia języka Fortran - FORmula TRANslation

- pierwszy kompilator: **1957**
- pierwszy oficjalny standard FORTRAN66: **1972**
- kolejny standard FORTRAN77: **1980**
- następnie Fortran90: **1991**

... rewolucyjne zmiany...

- kolejno Fortran95, Fortran2003, Fortran2008...

Skupimy się na instrukcjach wchodzących w skład standardu Fortranu 90, jednak z pewnymi ‘wycieczkami’ w stronę jego poprzednika, Fortranu 77. Kolejne rozszerzenia standardu, Fortran 95 nie były tak rewolucyjne.

Uwaga: przejście z F77 do F90 w praktyce przebiegało stopniowo: kompilatory wprowadzały liczne rozszerzenia, z których duża część znalazła się już ‘pełnoprawnie’ w standardzie F90.

FORTRAN77 jest całkowicie zawarty w Fortranie90.

Bardzo dużo podręczników elektronicznych, polecam linki ze strony:
http://www.icm.edu.pl/~aniat/fortran90/lektury.html

Obszerna informacja o języku na stronach Wikipedii:
http://en.wikipedia.org/wiki/Fortran

Uwaga: większość kompilatorów Fortranu oczekuje plików z rozszerzeniami `.f` lub `.for` (dla plików źródłowych napisanych w stałym formacie (*fixed-form*) lub rozszerzeniami `.f90` i `.f95` (dla plików źródłowych napisanych w formacie swobodnym).

Nowe możliwości wprowadzone przez Fortran 90

- swobodny format pisania programu;
- operacje na tablicach, funkcje wbudowane działające na tablicach;
- dynamiczne przydzielanie pamięci;
- przenoszalne typy danych (KIND);
- rekursja;
- moduły (MODULE);
- ulepszone (i rozszerzone) instrukcje sterujące;
- ......

Alfabet języka Fortran 90

Litery wielkie oraz małe:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Uwaga: Wielkie i małe litery nie są rozróżniane.

Cyfry:

```
0 1 2 3 4 5 6 7 8 9
```

Znaki specjalne:

spacja (odstęp)

```
‘ ” ( ) * + - / : = _ ! & $ ; < > % ? , .
```

Uwaga: Spacja ma znaczenie. Nie wolno umieszczać spacji w słowach kluczowych oraz nazwach. Kilka następujących po sobie spacji ma takie samo znaczenie jak jedna spacja.

Format pisania programu
FORTRAN77 – tzw. '*fixed-format*' charakteryzujący się następującymi cechami:
- treść instrukcji: 7-72 kolumna
- komentarze: `c` w pierwszej kolumnie
- etykiety: 1-5 kolumna
- znak kontynuacji: 'cokolwiek' w szóstej kolumnie linii będącej/będących kontynuacją
- długość nazw zmiennych i procedur: max 6 znaków

Fortran90 (*free-source form*)
- można pisać dowolnie (max 132 znaki w wierszu)
- ! – oznacza komentarz i może być umieszczony na początku wiersza, lub w dowolnym miejscu – wszystko w wierszu znajdujące się po wystąpieniu znaku ! jest traktowane jako komentarz
- nazwy – do 31 znaków; nazwy mogą zawierać 'podkreślnik' _
- kilka instrukcji w jednym wierszu – oddzielane średnikiem ;
- Kontynuacja linii – umieszczenie znaku & na końcu wiersza oznacza, że kolejna linia programu jest jego kontynuacją, max. 39 wierszy kontynuacji. Znak kontynuacji & działa jak spacja i nie można stosować go w środku nazw i słów kluczowych.
- W programie mogą znajdować się **puste wiersze**.

Nie powinno się tych dwóch formatów mieszać.

Struktura programu

<table>
<tr><td colspan="2">Instrukcja <code>PROGRAM, MODULE, SUBROUTINE, FUNCTION [nazwa]</code></td></tr>
<tr><td colspan="2">Instrukcja <code>USE</code></td></tr>
<tr><td colspan="2"><code>IMPLICIT NONE</code></td></tr>
<tr><td rowspan="2"></td><td>deklaracje</td></tr>
<tr><td>instrukcje</td></tr>
<tr><td colspan="2">Instrukcja <code>CONTAINS</code></td></tr>
<tr><td colspan="2">Procedury wewnętrzne</td></tr>
<tr><td colspan="2"><code>END { PROGRAM, MODULE, SUBROUTINE, FUNCTION } [nazwa]</code></td></tr>
</table>

Deklaracje, zmienne, stałe, typy
Typy zmiennych występujących w Fortranie90 dzielą się na trzy klasy:
- zmienne tekstowe
- zmienne logiczne
- zmienne numeryczne

Wyróżnia się sześć typów zmiennych

Fortran90	FORTRAN77

`CHARACTER :: lit  ! litera`
`CHARACTER (LEN=12) :: nazwisko`
`LOGICAL :: stan  !`
	`CHARACTER lit`
	`CHARACTER*12 nazwis`
	`LOGICAL stan`

`! wartosc .TRUE. lub .FALSE.`
`! typy numeryczne`
`REAL :: wysokosc`
`DOUBLE PRECISION  :: pi   ! 3.14...`
`INTEGER :: wiek  ! w latach`
`COMPLEX :: val  ! x + iy`
	`REAL wysoko`
	`INTEGER wiek`
	`COMPLEX val`

Ogólnie deklaracja ma postać:

```
< typ > [,< lista atrybutów >] :: < lista zmiennych, > [ =< wartość > ]
```

`< lista atrybutów >` zawiera listę atrybutów takich jak `PARAMETER`, `SAVE`, `INTENT`, `POINTER`, `TARGET`, **`DIMENSION`**, **`ALLOCATABLE`**. Obiektowi można przypisać kilka atrybutów.

To jest dobra okazja żeby zatrzymać się przy **tablicach**. Wcześniej jednak słowo na temat deklaracji `IMPLICIT`.

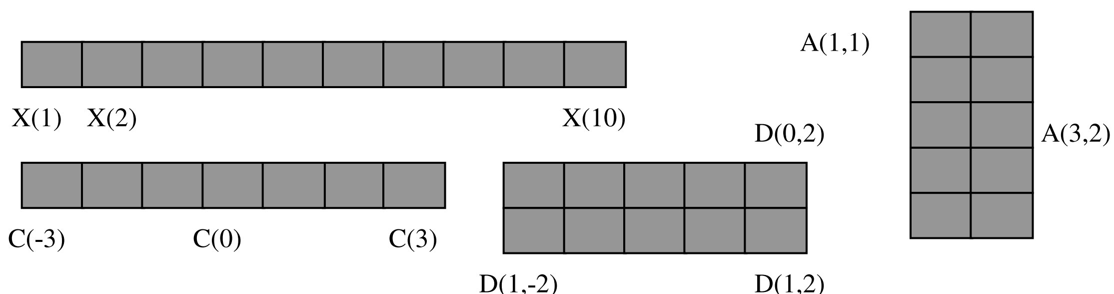
Fortran90 (podobnie jak FORTRAN77) **nie wymaga** deklarowania zmiennych i jest to niezmiernie niebezpieczna cecha!
Niezadeklarowana zmienna ma typ zależny od pierwszej litery w nazwie:
`REAL` A-H, O-Z, `INTEGER` I-N.
Deklaracja `IMPLICIT` pozwala zmienić tę konwencję. Ale nie to jest istotne. Najważniejszą deklaracją jest deklaracja `IMPLICIT NONE` – wielkie osiągnięcie standardu Fortranu 90.

Tablice

```
REAL, DIMENSION (10) :: X
INTEGER, DIMENSION (1:5, 1:2) :: A
REAL :: C(-3:3), D(0:1, -2:2)
```

Zadeklarowano cztery tablice:

- Tablica `X` o elementach rzeczywistych jest tablicą jednowymiarową (wektorem), bowiem podano jeden zakres indeksów, o 10 elementach **ponumerowanych od 1 do 10 (zapis `DIMENSION (10)` jest równoważny `DIMENSION (1:10)`;**
- Tablica `A` jest tablicą dwuwymiarową (macierzą), o pięciu wierszach i dwóch kolumnach.
- Tablica `C` jest wektorem o siedmiu elementach rzeczywistych
- Tablica `D` jest macierzą o dwóch wierszach i pięciu kolumnach.



Kolejność elementów w tablicy

Standard Fortranu 90 (w przeciwieństwie do standardu FORTRANU 77) nie definiuje w jaki sposób tablice są przechowywane w pamięci komputera.

Ale w sytuacjach, w których niezbędne jest zdefiniowanie uporządkowania (np. podczas wczytywania danych lub wypisywania wyników) obowiązuje kolejność znana z poprzednich wersji Fortranu: **elementy tablicy są uporządkowane w kolejności określonej najszybszą zmianą pierwszego indeksu**, co w przypadku tablic dwuwymiarowych (macierzy) oznacza, że **elementy są uporządkowane 'kolumnami'**.

Fortran90 wprowadził:
- bardzo dużo standardowych funkcji operujących na macierzach.
- tablice dynamiczne, których zakresy są określane podczas wykonania programu.

Deklaracja tablic dynamicznych
W deklaracjach tablic dynamicznych pomija się rozmiar tablic; wymiar tablicy definiuje się za pomocą dwukropków:

```
INTEGER, DIMENSION(:), ALLOCATABLE :: X
REAL, DIMENSION(:,:), ALLOCATABLE :: A
```

Uaktywnianie tablic dynamicznych
Przydzielenie miejsca w pamięci i uaktywnienie tablicy dynamicznej następuje poprzez wykonanie instrukcji:

```
ALLOCATE (X(10))
```

albo

```
ALLOCATE (X(10), STAT=ierr)
if (ierr /= 0) then
   PRINT*, 'blad allokacji tablicy X'
endif
```

Sprawdzenie statusu operacji (`STAT`) pozwala zareagować na ewentualny błąd wykonania alokacji pamięci. W przykładzie `ierr` jest zmienną typu `INTEGER`. Wartość `ierr` jest równa 0 gdy alokacja przebiegła pomyślnie; w przeciwnym przypadku jest różna od zera.

Zwalnianie pamięci
Zwolnienie miejca w pamięci następuje w wyniku wykonania instrukcji:

```
DEALLOCATE (X)
```

albo

```
if (ALLOCATE(X)) DEALLOCATE (X, STAT=ierr)
```

Wyrażenia arytmetyczne
Można wykonywać następujące operacje arytmetyczne (operatory):
- potęgowanie **
- mnożenie i dzielenie * /
- dodawanie i odejmowanie + -

Funkcje standardowe
- Omawianie wyrażeń arytmetycznych jest dobrym momentem do wspomnienia o funkcjach standardowych, nazywanych też funkcjami wbudowanymi (ang. *intrinsic*). Fortran 90 posiada 113 funkcji wbudowanych.

Standardowe funkcje matematyczne:
`sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), abs(x), exp(x), log(x), log10(x)`

Wyrażenia tekstowe
W wyrażeniu tekstowym mogą wystąpić następujące elementy (operandy):
- stałe i zmienne tekstowe
- podłańcuchy
- wywołania funkcji tekstowych

Operator konkatenacji oznacza się symbolem //. Konkatenacja polega na 'zlepianiu' ze sobą stałych lub zmiennych tekstowych występujących w wyrażeniu, np. wynikiem `'A'//'la'//' ma kota'` jest `'Ala ma kota'`.

Podłańcuchy (ang. *substring*)
Jeśli wartością zmiennej `tekst` jest '`Ala ma kota`' to:

```
tekst(5:6)                      ma
tekst( :3) (albo tekst(1:3)     Ala
tekst (8:) (albo tekst (8:11)   kota
tekst(:)                        Ala ma kota
```

Standardowe funkcje tekstowe

Wyrażenia relacji
Wyrażenia relacji służą do porównywania wartości dwóch wyrażeń arytmetycznych lub tekstowych. Wynik wyrażenia ma wartość logiczną `.TRUE.` gdy relacja zachodzi oraz `.FALSE.` w przeciwnym przypadku. Nie można porównać ze sobą wartości wyrażenia arytmetycznego i tekstowego.
Elementy wyrażenia relacji (operandami) są:
- wyrażenia arytmetyczne
- wyrażenia znakowe

Operatory relacji (operandy):
Dopuszczalne są dwa równorzędne sposoby zapisywania operatorów relacji. Jeden z nich, literowy, jest zapisem pochodzącym ze starych wersji języka Fortran; zapis za pomocą symboli matematycznych pojawił się w Fortranie90.

.GE.	**>=**	**większe niż**
.GT.	**>**	**większe równe**
.LE.	**<=**	**mniejsze równe**
.LT.	**<**	**mniejsze niż**
.NE.	**/=**	**nie równe**
.EQ.	**==**	**równe**

Na przykład wartością `5 .LT. 10` jest `.TRUE.`, zaś wartością `100.NE.100` jest `.FALSE.`

Wyrażenia logiczne
Wartość wyrażenia logicznego ma typ logiczny. Wyrażenie jest prawdziwe (`.TRUE.`) lub fałszywe (`.FALSE.`).

Wyrażenia logiczne konstruuje się w oparciu następujące operatory logiczne:

`.NOT.`	negacja
`.AND.`	koniunkcja
`.OR.`	alternatywa
`.EQV.`	tożsamość

`.NEQV.` nie tożsamość

Operatory zostały podane wkolejności zgodnej z ich priorytetem.

Instrukcje

Instrukcja podstawienia:

`zmienna = wyrażenie` (o odpowiednim typie)
W szczególności można wykonywać operacje na macierzach lub ich wycinkach.

Instrukcje warunkowe

Tylko jedna instrukcja wykonywana 'pod warunkiem':

```
IF (i>15) write (*,*) 'wartosc jest wieksza od 15'
```

Instrukcja blokowa warunkowa:

```
ala: IF (warunek) THEN
      instrukcja
      instrukcja
      ....
ELSE IF (warunek1) THEN
      instrukcja
      instrukcja
      ....
ELSE IF (warunek2) THEN
      instrukcja
      instrukcja
      ....
ELSE
      instrukcja
      instrukcja
      ....
ENDIF ala
```

Instrukcja SELECT CASE

Instrukcja `SELECT CASE` spełnia podobną rolę jak instrukcja warunkowa. Jest przydatna wtedy gdy algorytm wymaga wybrania jednej ze 'ścieżek postepowania' na podstawie wartości określonego wyrażenia.

Np.

```
SELECT CASE (i)         ! wyrażenie
   CASE (1, 5, 10)      ! zakres1 wartości wyrażenia
      instrukcja1
      instrukcja2
   CASE (20:)           ! zakres2 wartości wyrażenia
      instrukcja3
   CASE  DEFAULT        ! pozostałe zakresy
      instrukcja4
END SELECT

SELECT CASE (i)
      CASE (:0)
         print *, 'i  <= 0'
      CASE (2:5, 11:13)
         print*,'i z przedzialu [2,5] lub [11,13]'
      CASE (15, 17, 19)
         print*, 'i=15 lub i=17 lub i=19'
```

```
      CASE (20:)
         print*, ‘i>20’
      CASE DEFAULT
         print*, ‘i=1, 6 ,7,8,9,10,16 lub18’
END SELECT
```

Pętle – instrukcje cyklicznie powtarzane

W Fortranie 90 istnieje kilka różnych sposobów realizowania pętli (cyklu).

Instrukcja DO

Najprostsza pętla zapisuje się w Fortranie jako:

```
DO
   instrukcja1
   instrukcja2
   ......
END DO

ala: DO
   instrukcja1
   instrukcja2
   ......
END DO ala
```

Na przykład:

```
i = 0
DO
   i = i + 5
   print*, ‘i=’,i
   if ( i>10) EXIT
END DO
print *,’petla sie skonczyla. i ma wartosc’,i
```

Instrukcja DO-WHILE

Kolejna instrukcja cyklu, nazywana tutaj `DO-WHILE`, wymaga podania warunku determinującego jej wykonanie.

```
DO WHILE (warunek)
   instrukcja1
   instrukcja2
   ......
END DO
```

Indeksowana instrukcja DO

```
DO zmienna sterująca = wartość początkowa, wartość końcowa [, krok]
   instrukcja1
   instrukcja2
   ...
ENDDO
```

Liczba powtórzeń instrukcji cyklu wyraża się wzorem:

```
int ((wartość końcowa - wartość początkowa+krok) / krok )
```

Jeśli wynikająca z tego wzoru liczba powtórzeń jest równa bądź mniejsza od 0 to instrukcja cyklu nie zostanie wykonana ani razu.

```
DO i=0,9,1  ! albo równoważnie: DO i=0,9
   ....
   ....
```

```
END DO
```

albo

Fortran90	FORTRAN77

`DO i=1, n`
`   ....`
`   ....`
`END DO`
	`      DO 10 i=1, n`
	`         ....`
	`         ....`
	`10    CONTINUE`

Fortran90 stracił etykiety!!!

Przykład: Mnożenie macierzy

```
program mult_matrix
implicit none
real, dimension(5,5) :: a, b, c
integer :: i, j, k
real :: s
!
a = 1.
a(2,3) = 5.
b = 2.
b(:,1)=1.
! mnozenie tradycyjne
do i = 1, 5
  do j = 1, 5
    s = 0.
    do k = 1, 5
      s = s + a(i,k) * b(k,j)
    enddo
    c(i,j) = s
  enddo
enddo
!
write (*,*) 'wynik tradycyjny'
do i = 1, 5
  write (*,100) (c(i,j), j=1,5)
enddo
!
c = matmul(a, b)
write (*,*) 'wynik automatyczny'
do i = 1, 5
  write (*,100) (c(i,j), j=1,5)
enddo
100 format (5(2x, f6.2))
end program mult_matrix
```

Wyniki:

```
wynik tradycyjny
    5.00   10.00   10.00   10.00   10.00
    9.00   18.00   18.00   18.00   18.00
    5.00   10.00   10.00   10.00   10.00
    5.00   10.00   10.00   10.00   10.00
    5.00   10.00   10.00   10.00   10.00
 wynik automatyczny
    5.00   10.00   10.00   10.00   10.00
    9.00   18.00   18.00   18.00   18.00
```

```
    5.00   10.00   10.00   10.00   10.00
    5.00   10.00   10.00   10.00   10.00
    5.00   10.00   10.00   10.00   10.00

Press RETURN to close window . . .
```

Podprogramy

Fortran wyróżnia dwa rodzaje podprogramów: procedury (`SUBROUTINE`) oraz funkcje (`FUNCTION`).

W zależności od ich umiejscowienia w strukturze programu w Fortranie 90wyróżnia się trzy kategorie podprogramów:

- podprogramy wewnętrzne,
- podprogramy w modułach,
- podprogramy zewnętrzne.

- Podprogram jest podprogramem wewnętrznym jeśli znajduje się pomiędzy instrukcją `CONTAINS` a instrukcją `END` (`END PROGRAM`, `END MODULE` lub `END` podprogramu zewnętrznego).
- Podprogram wewnętrzny jest podprogramem należącym do modułu jeśli znajduje się pomiędzy instrukcją `CONTAINS` oraz instrukcją `END MODULE`.
- Podprogram jest podprogramem zewnętrznym jeśli nie znajduje się wewnątrz segmentu `PROGRAM`, `MODULE` lub innego podprogramu. Podprogram zewnętrzny może znajdować się w oddzielnym pliku lub w tym samym pliku co inne segmentu po instrukcji `END` (`END PROGRAM`, `END MODULE` lub `END` należący do innego podprogramu).

Budowa podprogramu typu `SUBROUTINE` jest następująca:

```
SUBROUTINE nazwa_procedury (arg1, arg2, ..., argn)
IMPLICIT NONE
deklaracje
instrukcje
podprogramy_wewnętrzne
END SUBROUTINE nazwa_podprogramu

Wywołanie:
CALL nazwa_procedury (arg1, arg2, ...., argn)
```

Budowa podprogramu typu FUNCTION jest następująca:

```
typ FUNCTION nazwa_funkcji (arg1, arg2, ..., argn)
IMPLICIT NONE
deklaracje
instrukcje
podprogramy_wewnętrzne
END FUNCTION nazwa_funkcji
```

W odróżnieniu od podprogramu typu `SUBROUTINE`, w którym nazwa służy jedynie do identyfikacji, nazwa podprogramu typu `FUNCTION` oprócz roli identyfikacyjnej pełni też funkcję 'nośnika' wyniku - stąd konieczność przypisania funkcji **typu**.

Wywołanie: poprzez umieszczenie w wyrażeniu.

Atrybut `INTENT`

Atrybut `INTENT` umożliwia określenie roli spełnianej przez poszczególne parametry formalne:

- `INTENT (IN)` - parametr jest przeznaczony do dostarczenia danych. Próba zmiany wartości parametru w obrębie podprogramu jest błędem.
- `INTENT (OUT)` - argument służy do wyprowadzenia wyniku.
- `INTENT (INOUT)` - argument może pełnić rolę zarówno danej jak i wyniku. Wartość parametru może ulegać zmianie w toku realizacji podprogramu. Jest to domyślna wartość.

Przykład:

```
SUBROUTINE SUMA (a, b, c)
IMPLICIT NONE
REAL, INTENT (IN) :: a,b
REAL, INTENT (OUT)::c
c=a+b
END SUBROUTINE SUMA
```

Deklaracja `INTERFACE`

Zabezpieczeniem zgodności parametrów formalnych oraz parametrów aktualnych jest 'zadeklarowanie' podprogramu w segmencie go wywołującym. Służy do tego 'łącznik' `INTERFACE`, w którym podaje się podstawowe informacje o podprogramie i jego parametrach. Łącznik rozpoczyna się słowem kluczowym `INTERFACE`, zaś kończy frazą `END INTERFACE`. Jego treść składa się z nagłówka podprogramu, deklaracji parametrów oraz instrukcji `END SUBROUTINE nazwa_procedury` lub `END FUNCTION nazwa_funkcji`. Łącznik `INTERFACE` umieszcza się bezpośrednio po deklaracji `IMPLICIT NONE`, a przed pozostałymi deklaracjami. Użycie `INTERFACE` nie jest wymagane, ale bardzo zalecane.

Przykład:

```
PROGRAM SUMOWANIE
IMPLICIT NONE
INTERFACE
   SUBROUTINE SUMA(a,b,c)
   REAL, INTENT(IN) ::a, b
   REAL, INTENT(OUT)::c
   END SUBROUTINE SUMA
END INTERFACE
REAL:: x=2.,y=3.,z
CALL SUMA(x,y,z)
WRITE(*,*) 'suma=', z
CALL SUMA(20., 10., x)
WRITE(*,*) 'suma=', x
END PROGRAM SUMOWANIE
```

Przykład:

```
PROGRAM PROSTY
IMPLICIT NONE
REAL :: AA=2., BB=3., CC
CALL SUMA (AA,BB,CC)
WRITE (*,*) 'SUMA WYNOSI ',CC
CONTAINS
  SUBROUTINE SUMA (a, b, c)
  REAL, INTENT (IN) :: a,b
  REAL, INTENT (OUT)::c
  c=a+b
  END SUBROUTINE SUMA
END PROGRAM PROSTY
```

Uwagi:

- Podprogramy wewnętrzne mogą być wywoływane tylko z segmentu, w którym się znajdują.
- Mają automatycznie dostęp do wszystkich zmiennych, zadeklarowanych w segmencie głównym (dlatego też, między innymi, nie ma potrzeby umieszczać w nich deklaracji `IMPLICIT NONE`).
- Podprogram może zawierać odwołania do innych podprogramów należących do tego samego programu głównego, dołączonych modułów oraz podprogramów zewnętrznych.
- Podprogram wewnętrzny nie może zawierać w sobie innego podprogramu.

Moduły

Fortran 90 definiuje moduły jako zbiór deklaracji i podprogramów. Każdy moduł składa się z jednej lub dwóch składowych; te składowe to część zawierająca specyfikacje (deklaracje), takie jak bloki `INTERFACE`, deklaracje `IMPLICIT, PARAMETER, TYPE`. W szczególności w tej części może pojawić się wywołanie innego modułu `USE nazwa_modułu`. W drugiej części modułu (o ile występuje) umieszcza się kody podprogramów.
Moduł rozpoczyna się instrukcją:

```
MODULE nazwa
```

a kończy się instrukcją:

```
END MODULE nazwa
```

Przykład:

```
MODULE single
   INTEGER, PARAMETER :: float = selected_real_kind (6, 37)
   REAL (foat) :: pi = 3.1415927_float
   REAL (float) :: e = 2.7182818_float
END MODULE single

MODULE double
   INTEGER, PARAMETER :: float = selected_real_kind (15, 307)
   REAL (foat) :: pi = 3.141592653589793_float
   REAL (float) :: e = 2.718281828459045_float
END MODULE double
```

W zależności od potrzebnej precyzji obliczeń, wykorzystuje się jeden z dwóch modułów:

```
PROGRAM constants
   USE single
   PRINT *, ‘constants: pi’, pi, ‘ oraz e’, e
   CALL more_precision
END program constants

SUBROUTINE more_precision
   USE double
   PRINT *, ‘constants: pi’, pi, ‘ oraz e’, e
END SUBROUTINE more_precision
```

Uwaga: instrukcja `USE nazwa_modułu` nie może wystąpić w kontekście instrukcji warunkowej.

KIND – parametryzowana precyzja typów

Precyzja oraz zakres wartości danych o określonym typie na jednym procesorze niekoniecznie jest taki sam na innym, co może być przyczyną trudności z przenoszalnością (*portability*)

programów. Koncepcja `KIND` wprowadzona w standardzie Fortranu 90 jest mechanizmem umożliwiającym parametryzację typów danych za pomocą całkowitej wartości `KIND`.
Całkowita wartość `KIND` określa model reprezentowania wartości danego typu, czyli precyzję (liczbę cyfr znaczących) oraz zakres wartości. W szczególności możliwe jest definiowanie różnej precyzji (lub reprezentacji) dla zmiennych typu `INTEGER, REAL` oraz `COMPLEX`.

Wykorzystuje się następujące funkcje wbudowane:

`KIND(x)` – zwraca parametr charakteryzujący sposób zapisu zmiennej `x`
`R = RANGE(x)` – zwraca liczbę całkowitą określającą zakres zmiennej `x`, rozumiany jako wartość maksymalna `10**R` oraz najmniejsza liczbę blisko zeru `10**(-R)`.
`P = PRECISION(x)` – zwraca liczbę cyfr znaczących w reprezentacji zmiennej `x`
`K = SELECTED_REAL_KIND( P, R)` – zwraca parametr charakteryzujący zmienną o `P` cyfrach znaczących i zakresie wartości `R`
`K = SELECTED_INT_KIND (R)` – zwraca parametr charakteryzujący zmienną całkowitą przyjmującą wartości z przedziału `-10**R - 10**R`

Przykład:

```
program rodzaj
integer, parameter :: k=selected_real_kind(15,99)
real :: x=5.
real (kind=k) :: long=3.
double precision :: d=10.
integer :: i=2
write (*,*) 'int:',range(i),kind(i)
write (*,*) 'real:',range(x),precision(x),kind(x)
write (*,*) 'double:',range(d),precision(d),kind(d)
write (*,*) 'long'
write (*,*) 'real:',range(long),precision(long),kind(long)
end program rodzaj
```

Wyniki:

```
int:           9          3
real:          37          6           1
double:         307          15           2
long
real:         307          15          2

Press RETURN to close window . . .
```

Zmienne (i stałe) `DOUBLE PRECISION` nie posiadają zróżnicowanej dokładności i dlatego ich wykorzystanie nie jest zalecane. Zamiast `DOUBLE PRECISION` zaleca się wykorzystanie odpowiednio sparametryzowanego typu `REAL`.

Obiekty o różnych wartościach `KIND` mogą być mieszane w wyrażeniach, reprezentacja (dokładność) wyniku podlega regułom. Natomiast jest wymagane, aby argumenty podprogramów zgadzały się zarówno co do typu jak i rodzaju `KIND`! Przy przekazywaniu parametrów konwersja nie zachodzi!
Dlatego tym bardziej jest dobrą praktyką inicjalizowanie parametrów `KIND` w modułach, z których następnie korzystają inne segmenty programu.