Fortran 90/95
wykład 1

Janusz Andrzejewski

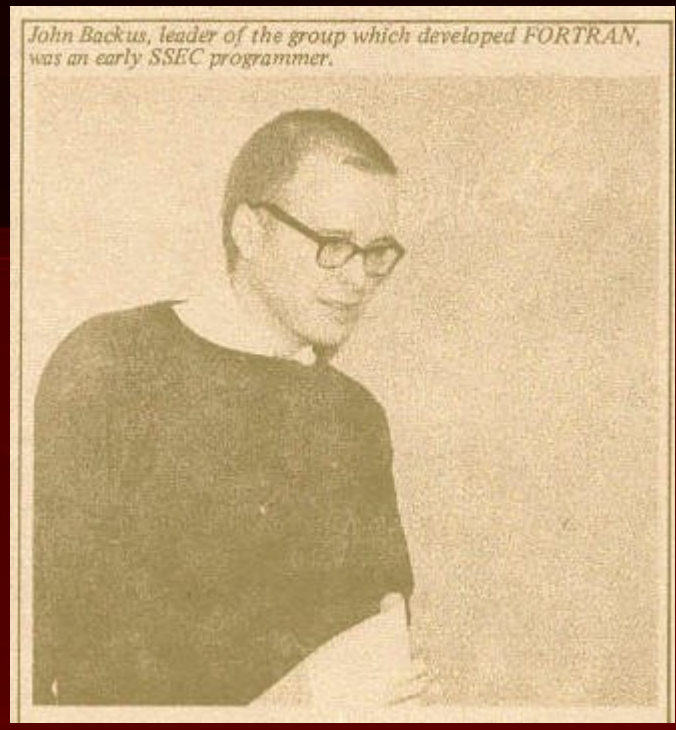
13/11/11

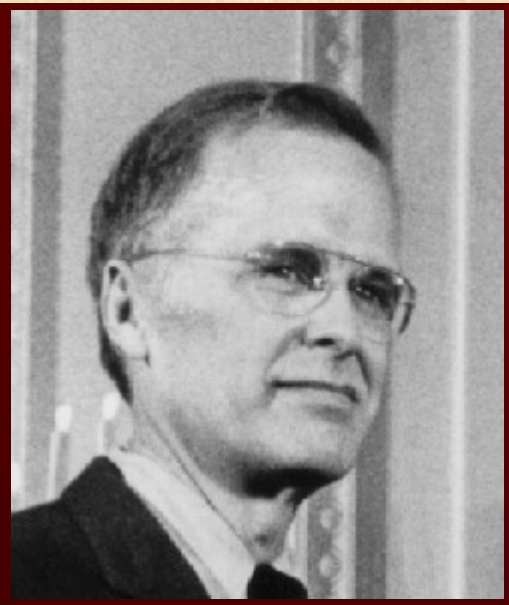
PLAN

- Historia
- Co złego w FORTRANie 77
- Co nowego w Fortranie 90/95
- Nowy styl
- Typy danych, deklaracja zmiennych i stałych
- Struktury danych
- Tablice
- Przekazywanie argumentów

Historia – FORTRAN I

- 1954-57 – jako jeden z **pierwszych** języków wysokiego poziomu w laboratorium IBM pod kierownictwem John'a Backusa powstał FORTRAN I (akronim od **FOR**mula **TRAN**slation) – jako język który w łatwy sposób pozwalał zapisywać i obliczć matematyczne formuły. Był **pierwszym** językiem który posiadał kompilator. Dzięki niemu można było 5 razy szybciej pisać programy, przy okolo 20% spowolnienu w porównaniu z programem napisanym w języku maszynowym. FORTRAN I przez 20 lat posiadał najlepszy kompilator. W 1977 John Backus otrzymał nagrodę TURINGA za: *„ głęboki, wpływowy i trwały wkład do projektowania praktycznych systemów programowania wysokiego poziomu, w szczególności poprzez pracę nad FORTRAN'em i za przełomową publikację nt. formalnych procedur specyfikacji języków programowania”*





John Backus (3.12.1924-17.03.2007)

Historia

Skład grupy tworzącej kompilator FORTRANu (zdjęcie z 1982r). W skład wchodzili: inżynierowie, naukowcy, matematycy, szyfrant, szachista

Historia – złote lata

- 1957 – FORTRAN I – zawierał 32 instrukcje
- 1958 – FORTRAN II – znacząco udoskonalony, pozwalał na osobną kompilację modułów. Dodano możliwość pisania procedur oraz funkcji. Później dodano także typ DOUBLE PRECISION oraz COMPLEX
- 1958 – Fortran III – nigdy publicznie nie opublikowany, posiadał możliwość wstawiania kodu maszynowego.
- 1961 – Fortran IV – dalsze udoskonalenia, dodatnie COMMON oraz EQUIVALENCE. Dodano typ LOGICAL oraz stworzono możliwość budowania wyrażeń logicznych. Usunięto instrukcje niskopoziomowe. Wersja ta stała się nieformalnym standardem języka.

Historia – złote lata

- Maj 1962 – ASA (*American Standards Association* teraz znana jako *American National Standards Institute*) zaczeła pracę na standaryzacją języka
- 1966 – FORTRAN 66 – **pierwszy** standard języka został opublikowany. Był to też **pierwszy** jakikolwiek standard dotyczący języków programowania.
  - Wersja FORTRAN – oparta na wersji FORTRAN IV
  - Wersja Basic FORTRAN – oparta na wersji FORTRAN II

Historia - stagnacja

- Przełom lat 60 i 70 – FORTRAN przestał być w awangardzie, nie wytyczał/adoptował nowych idei jak tzw. programowanie strukturalne
- 1978 – FORTRAN 77 (w skrócie F77)– nowy standard języka (norma ANSI X3.9-1978)
  - Pętla DO mogła posiadać malejący indeks pętli
  - Blokowa instrukcja IF ... THEN ... ELSE ... ENDIF. Przed F77 była tylko IF GOTO instrukcja – powodująca tzw. spaghetti program
  - pre-testowanie pętli DO. Przed F77 pętle DO zawsze się przynajmniej raz wykonywały, co powodowało konieczność użycia instrukcji IF GOTO jeśli chciało się mieć oczekiwane zachowanie
  - Nowy typ CHARACTER. Przed F77 znaki były przechowywane w zmiennych typu INTEGER
  - Apostrof mógł zaczynać i kończyć łańcuch znaków
  - Główny segment mógł się kończyć bez instrukcji STOP

Historia - stagnacja

- 1978 – rozszerzona wersja F77 opublikowana, rozszerzenie MIL-STD-1753
  - Dodano instrukcje DO WHILE oraz END DO
  - Dodano instrukcje INCLUDE
  - Dodano IMPLICIT NONE (wariant instrukcji IMPLICIT)
  - Dodano operacje na bitach
- 1980 – międzynarodowy standard, norma ISO 1539-1980

Co złego w FORTRAN'ie 77

- Nie ma dynamicznych struktur danych
- Nie ma możliwości zdefiniowania typów danych
- Nie ma wbudowanych struktur danych
- Nie ma kontroli nad poprawnością przekazywania argumentów
- Mała gama instrukcji sterujących, powodujących powstawanie tzw. spagetti programu
- Archaiczność kodu
  - Sztywny format zapisu programu
  - Tylko duże litery
  - Zmienne mogą mieć długość 6 znaków

Co nowego w Fortranie 90

- Wolny styl pisania programów
- Tablice – operacje na tablicach stają się podstawą języka
- Dynamiczne struktury danych
- Możliwość definiowania własnych struktur danych, przeciążania operatorów lub definiowania własnych
- **MODULE** – nowy sposób grupowania zmiennych, funkcji i procedur
- Nowe instrukcje kontrolujące przebieg programu
- Możliwość używania rekurencji
- Możliwość używania opcjonalnych argumentów
- ...

Nowy wygląd

- Małe litery mogą być używane, jednak Fortran jest nieczuły na wielkość liter
- Nazwy zmiennych mogą mieć długość 31 znaków, znakami mogą być także znak podkreślenia oraz cyfry
- W jednej linii może być kilka instrukcji oddzielonych średnikiem

```
suma=0.0;  iloczyn=0.0; iloraz=0;
```

- Komentarz może zaczynać się w dowolnym miejscu lini

```
Suma_czesciowa=0.0  ! pomocnicza zmienna
```

- Stałe znakowe mogą być zamknięte poprzez pojedynczy jak i podwójny cudzysłów

```
write(*,*)”It isn't broke. Don't fixed it”
```

Wolny styl pisania programów

- Instrukcje mogą pojawić sie w dowolnym miejscu linii, linie mogą mieć 132 znaki długości
- Znak **„!”** służy do zaznaczania komentarza
- Znak **„&”** służy jak znak kontynuacji linii

```
call moja(a, b  &   ! najwazniejsze argumenty
                c, d)   ! mniej ważne argumenty
```

- Nie w każdym miejscu można użyć komentarzy

```
 write(*,*)”To jest lancuch &
                & i jego dalsza czesc” ! Miedzy tymi liniami nie
                ! nie moze byc komentarza
```

- Spacje są istotne

```
  liczba=1 000 000  ! Poprawnie tylko dla sztywnego stylu
```

Program „Hallo world”

```
PROGRAM  pierwszy
   WRITE(*,*)'Czesc'
END PROGRAM pierwszy
```

Każdy segment składa się z 2 części

- Bloku instrukcji biernych (BIB)
- Bloku instrukcji czynnych (BIC)

Przed BIB jest nagłówek (np. **PROGRAM**), koniec segmentu zakończony jest słowem kluczowym **END**

Typ całkowity

Standartowy sposób (F77):

INTEGER

Nowy sposób określania typu całkowitego (F90)

selected_int_kind(r)

Funkcja zwraca tzw. kind wartość (o domyślnym typie integer) taką, że typ ten może reprezentować wszystkie liczby całkowite z zakresu -10^r < n < 10^r. W przypadku kiedy wiele typów całkowitych może reprezentować dany zakres, wybierany jest najmniejszy typ. W przypadku gdy nie ma takiego typu który był by w stanie reprezentować wszystkie liczby całkowite, funkcja zwraca wartość „-1”

Przykład

```
INTEGER    nn, na
integer, parameter :: k6=selected_int_kind(6)
integer(k6)  :: i, j;
integer(2) :: l, m    ! ryzykownie
integer(kind=k6) :: k
! Sposoby deklaracji stałych całkowitych
    23
    -456
  23_k6  ! deklaracja stałej liczbowej o typie określonym przez
          ! kind value wynoszaca k6
  kind(1)  - funkcja zwracająca kind value dla domyślnego typu
             całkowitego
   kind(2_k6)  - wartość kind dla typu o kind=k6
```

Typ zmiennoprzecinkowy

Standartowy sposób (F77)

REAL, DOUBLE PRECISION

Do określania typów zmiennoprzecinkowych używamy:

selected_real_kind(p, r)

Funkcja ta zwraca liczbę całkowita (o domyślnym typie całkowitym) której wartością jest typ (kind value) jaki ma liczba rzeczywista o precyzji określonej przez argument p i o zakresie określonej przez argument r. Przynajmniej jeden z argumentów musi być podany. W przypadku gdy nie ma odpowiedniego typu danych zwracana jest:

- -1 – gdy nie ma odpowiedniej precyzji
- -2 – gdy nie ma odpowiedniego zakresu
- -3 – gdy nie ma ani zakresu ani precyzji

Przykład

```
integer, parameter ::long=selected_real_kind(9, 99)
  DOUBLE PRECISION   dx, dy
  REAL         rx, ry
  real :: a, b
  real(long)::  c, d
  real(8)  :: x, y  ! Ryzykownie, najprawdopodobniej jest to double
                    !precision


   A=1.0
  c=2.0_long
  X=4.0_8   ! ryzykownie
```

Typ COMPLEX

Typ COMPLEX składa się z 2 licz rzeczywistych, więc do określenia typu COMPLEX używa sie tego samego sposobu co do określenia typu dla licz rzeczywistych

```
 COMPLEX  cz

 DOUBLE COMPLEX  dz, zz

integer, parameter :: lzsp=selected_real_kind(9, 99)

complex :: z1

complex(lzsp) ::  z2


  Z1=(1.0, 2.0)

  z2=(1.0_lzsp, 3.0_lzsp)
```

Struktury danych

W Fortranie 90/95 można definiować własne nowe struktury danych.

Postać:

```
type nazwa_SD
    typ_zmiennej :: nazwa_zmiennej
     ...
end type nazwa_SD
```

Przykład:

```
type osoba
    character(len=20) :: imie
    real  :: wiek
    integer :: id
end type osoba
```

Struktury danych

Deklaracja zmiennych o własnym typie danych

```
type(osoba)  :: me
```

Operacje na strukturach danych.

Dostęp do poszczególnych pól SD poprzez operator **„%”**

```
me%imie=”Janusz”
me%wiek=podaj_wiek()
me%id+9   ! jako wyrażenie
```

Tablice

Deklaracja tablic

```
REAL :: tablica(2, 3), tab0(-2:3, -3:4), scalar, wektor(1234)
REAL, DIMENSION(20, 30) :: tab1, tab3, tabliczka
REAL, DIMENSION(-10:10, -20:20) :: tabX, tabY
! Nadawanie tablicy poczatkowej wartosci
 INTEGER :: options(3)=(/1, 2, 3/)
 CHARACTER(LEN=2), PARAMETER:: dzien(0:6)=(/'PN', &
                              'WT','SR','CZ','PT','SB','ND'/)
```

W Fortranie 90/95 tablice mogą posiadać do 7 wymiarów

Tablice

```
REAL :: X(2, 5, -1:8)
```

Terminologia:

Dla powyżej zadeklarowanej tablicy:

- Wymiar tablicy wynosi 3 (rank)
- Zakres tablicy wynosi 2, 5 oraz 10 (extents)
- Kształt tablicy wynosi (/2, 5, 10/) (shape)
- Rozmiar tablicy wynosi 100 (size)

Przekazywanie argumentów

W Fortranie 90/95 jest możliwość dookreślenia sposobu przekazywania argumentów do procedur i funkcji. W deklaracji argumentów formalnych, należ podać dodatkowy atrybut INTENT z odpowiednimi opcjami

- Atrybut INTENT(IN) – argument formalny na wejściu do segmentu ma ustaloną wartość, wartości tego argumentu (zmiennej) nie można i nie wolno zmieniać
- Atrybut INTENT(OUT) – argument formalny na wejściu do segmentu nie ma ustalonej wartości, w segmencie można i należy ustalić wartość tego argumentu
- Atrybut INTENT(INOUT) – argument formalny na wejściu do segmentu ma ustaloną wartość oraz w segmencie wartość tego argumentu można zmieniać

Przykład

```
SUBROUTINE oblicz(a, wym, b)
INTEGER, INTENT(IN) :: wym
REAL, INTENT(IN), DIMENSION(:) :: a
REAL, INTENT(OUT) :: b
INTEGER  :: i
b=0.0
DO i=1, wym
   b=b+a(i); END DO
END SUBROUTINE oblicz
```

```
Program test
 INTEGER, PARAMETER :: wym=20
 REAL  :: a(wym), b
 INTEGER  i
DO i=1, wym; a(i)=0.0+i; END DO
CALL oblicz(a, wym, b)
END
```

Przykład (2)

```
SUBROUTINE ktoryraz(k)
 INTEGER, INTENT(INOUT) :: k
  k=k+1
  Write(*,*)'Wywolales &
&procedurke ',k,' razy'
END SUBROUTINE
```

```
Program dwa
  INTEGER :: i, k
K=0
DO i=1, 10
   CALL ktoryraz(k)
End do
END PROGRAM dwa
```

Dziękuję za uwagę