

Programowanie w języku Fortran

dr inż. Maciej Woźniak ¹

¹Katedra Informatyki, Wydział Informatyki, Elektroniki i Telekomunikacji,
Akademia Górniczo-Hutnicza, Kraków, Polska

- macwozni@agh.edu.pl
- D17, 2.13
- Konsultacje - TBA
- dodatkowe terminy konsultacji po uzgodnieniu mailowym
- <https://github.com/macwozni>
- home.agh.edu.pl/~macwozni/fort

Sposób zaliczenia

- Obowiązkowa obecność na laboratorium
- Maksymalnie jedna obecność nieusprawiedliwiona
- 3 zadania domowe po 40 pkt.
- Oceniana aktywność na zajęciach
- Suma punktów = $\min(100, \text{suma punktów za zadania i aktywność})$

Aktualnie istniejąca literatura dla Fortran-a w wersji 2008:

- Walter S. Brainerd, "Guide to Fortran 2008 Programming"
- Ian Chivers, Jane Sleightholme, "Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77"
- Arjen Markus, "Modern Fortran in Practice"
- Michael Metcalf, John Reid, Malcolm Cohen, "Modern Fortran Explained"

- **gfortran** (pakiet gcc)
- **ifort** (kompilator Intel-a, komercyjny)
- pgf90 (Portland Group Inc., komercyjny + wersja community)
- flang (LLVM)
- nagfor (NAG, komercyjny)
- FTN95 (SilverFrost, komercyjny, kompiluje do maszyny .NET)
- IBM XL (IBM, komercyjny)
- ftn (CRAY, komercyjny)

De facto wspierają kolorowanie składni i ewentualnie podstawowe podpowiadanie.

- vim
- Emacs
- Kate/KWrite
- NetBeans
- Eclipse
- IntelliJ
- Atom
- Visual Studio Code

- przed FORTRAN77 (*.f, *.F)
- FORTRAN77 (*.f, *.F)
- Fortran90 (*.f90, *.F90)
- Fortran95 (*.f90, *.F90)
- Fortran2003 (*.f90, *.F90)
- Fortran2008 (*.f90, *.F90)
- Fortran2018 (*.f90, *.F90)

gfortran

- `-ffree-form -std=f2008 -fimplicit-none -cpp -Wall -pedantic`
- `-fbounds-check`
- `-O3 -pthread -funroll-all-loops`
- `-O0 -g3 -fbacktrace -fcheck=all -fdump-fortran-optimized`

ifort

- `-std08 -module . -implicitnone -fpp -pedantic`
- `-check bounds`
- `-O3 -funroll-all-loops`
- `-O0 -g -traceback -check all -debug extended
-debug-parameters all`

Prosty program

Kod programu znajduje się pomiędzy słowami kluczowymi **program** oraz **end**. Poniżej są trzy równoważne sposoby zakańczania programu.

```
program main  
    <code here>  
end
```

```
program main  
    <code here>  
end program
```

```
program main  
    <code here>  
end program main
```

Podstawowe typy danych

❶ integer

- `integer :: i`
- `integer(kind = 4) :: j`

❷ float

- `real :: a`
- `real(kind = 4) :: b`

❸ boolean

- `logical :: ok`

Słowo kluczowe **kind** określa precyzję.

❶ pojedyncza

- `kind = 4`
- `kind = 1`

❷ podwójna

- `kind = 2`
- `kind = 8`

❸ poczwórna

- `kind = 3`
- `kind = 16`

Domyślną wartością `kind` (o ile nie poda się inaczej) jest 4. Wartość `kind` można zmienić podając ją jawnie w kodzie, albo przez odpowiednią opcję kompilatora.

Odpowiedni typów zmiennych z *C* w Fortran

- *integer* - **integer (kind=4)**
- *long* - **integer (kind=8)**
- *float* - **real (kind=4)**
- *double* - **real (kind=8)**

Typy danych implicit

W Fortran-ie standardowo o typie danej (o ile nie zadeklarujemy inaczej) decyduje pierwsza litera zmiennej. Przykładowo zmienna zaczynająca się na “i” (np. `iter=0`) jest integer. Natomiast zmienna zaczynająca się na “a” (np. `atom=0.d0`) jest real.

In Fortran GOD is REAL (unless declared INTEGER)

Używanie zmiennych typu implicit jest źródłem wielu błędów. Dlatego zalecam słowo kluczowe **implicit none**.

```
program main
implicit none
<declare variables>
    <code here>
end program
```

Pętle w Fortran-ie liczą do podanych wartości włącznie. W poniższym przykładzie od 1 do iSIZE włącznie.

```
do i=1,iSIZE  
    <code here>  
end
```

```
do i=1,iSIZE  
    <code here>  
end do
```

```
do i=1,iSIZE  
    <code here>  
enddo
```

Domyślnie pętle inkrementują o 1. W przypadku, gdy potrzeba wykonywać inkrementację o inną wartość bądź dekrementację - podaje się trzeci parametr do deklaracji pętli.

```
do i=1,iSIZE  
    <code here>  
end do
```

```
do i=1,iSIZE,10  
    <code here>  
end do
```

```
do i=1,iSIZE,-1  
    <code here>  
end do
```

“Routine”

Pewnym odpowiednikiem funkcji zwracających **void** z C są **subroutine**. Nie zwracają one eksplicite wartości. Przyjmują one parametry, które mogą modyfikować.

```
subroutine mm ()  
    <code here>  
end
```

```
subroutine mm ()  
    <code here>  
end subroutine
```

```
subroutine mm ()  
    <code here>  
end subroutine mm
```


“Routine”

Typy zmiennych, które przyjmują routiny (i funkcje) deklarujemy wewnątrz bloku routiny/funkcji

```
subroutine mm (a,b)
implicit none
real (kind=4) :: a,b
    <code here>
end subroutine
```

Funkcja w przeciwieństwie do “routine” zwraca wartość. Nie występuje tutaj odpowiednik **return**. Zamiast tego przypisuje się zwracaną wartość do zmiennej zadeklarowanej jako **result**.

```
function lerp (a,b) result (val)
implicit none
real (kind=4) :: a,b,val
    <code here>
    val = <value>
end function
```

Opcjonalnie przypisuje się zwracaną wartość do zmiennej zadeklarowanej jako nazwa funkcji.

```
function lerp (a,b)
implicit none
real (kind=4) :: a,b,lerp
    <code here>
    lerp = <value>
end function
```

Zmienne podawane w wywołaniu mogą być przekazywane tylko w jedną lub obie strony. Domyślnym **intent** jest **inout**.

Trzy typy **intent**:

- **in** - wartość przekazywana tylko do funkcji, nie może być wewnątrz niej modyfikowana
- **out** - wartość przekazywana tylko na zewnątrz funkcji, nie odczytuje wartości zewnętrznej
- **inout** lub **in out** - wartość przekazywana w obie strony

Intent

Funkcja i odpowiadająca jej “routine”.

```
function lerp (a,b) result (val)  
implicit none  
real (kind=4), intent(in) :: a,b  
real (kind=4) :: val  
    <code here>  
    val = <value>  
end function
```

```
subroutine mm (a,b, val)  
implicit none  
real (kind=4), intent(in) :: a,b  
real (kind=4), intent(out) :: val  
    <code here>  
    val = <value>  
end subroutine
```

Wywoływanie powyższych funkcji i routine.

```
result = lerp(a,b)
```

```
call mm (a,b, result)
```

Domyślnie tablica jest indeksowana od 1 do iSIZE włącznie. Tablice mogą mieć dowolną ilość wymiarów.

```
integer :: array1(iSIZE)
```

```
integer :: array2(iSIZE,iSIZE)
```

```
integer :: array3(iSIZEX,iSIZEY,iSIZEZ)
```

Na następnych wykładach

Na najbliższych wykładach

- więcej o typach danych
- rzutowanie typów
- instrukcje warunkowe
- makra i kompilacja warunkowa
- więcej o tablicach (w tym inne sposoby indeksowania)
- slice-y tablic i operacje na tablicach/podtablicach
- parametry z linii poleceń
- moduły