Programowanie w języku Fortran

dr inż. Maciej Woźniak [1]

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

Pointer vs allocatable

Allocatable

- wymagają jawnego **allocate**
- kompilator potrafi je automagicznie zdealokować na wyjściu z procedury (kiedy?)

Pointer

- bardziej swobodne użycie
- mogą wskazywać na pojedyncze (nieciągłe) fragmenty tablic
- mogą wskazywać na statyczne fragmenty pamięci

Move Alloc

Przealokowanie tablicy

```
real, allocatable :: arr, tmp
allocate(arr(50))
...
allocate(tmp(100))
tmp(1:50) = arr
deallocate(arr)
call move_alloc(tmp, arr)
```

Static - save

Odpowiednikiem **static** jest **save**. Zmienne oznaczone jako **save** będą zachowywać swoje wartości pomiędzy wywołaniami funkcji/subroutiny.

UWAGA

Nie należy używać **save** do całego modułu/funkcji/subroutiny, czyli bez podawania zmiennej!

```
subroutine mysub ()
  integer, save :: i
```

Static - save

Poniżej są przykłady niezalecanego użycia

```
subroutine mysub ()
  save
  integer :: i
```

```
module mymod
  save
  integer :: i
```

Dlaczego wykonanie poniższego kodu może zakończyć się błędem?

```
integer :: i(50)
if( (size(i).GE.60) .AND. (i(60).EQ.1) ) then
   write(*,*) i(70)
endif
```

```
!-----------------------------------------------------------------
!> @author
!> Maciej Wozniak
!
! DESCRIPTION:
!> Allocates and fills the knot vector on \f$ [0, 1] \f$.
!>Number of subintervals is \f$ N = n-p+1 \f$.
!> \f$ 0 \f$ and \f$ 1 \f$ are repeated \f$ (p+1) \f$ times.
!
! REVISION HISTORY:
! 21 11 2017 - Initial Version
!
!> @param[in] n - number of functions on the knot minus one
!> @param[in] p - degree of polynomial
!> @param[out] U - array to fill with points
!> @param[out] nelem - number of elements
!-----------------------------------------------------------------
```

f2py

Łatwo jest zawołać subroutiny Fortrana z Pythona. Przydatnym narzędziem jest **f2py**, który generuje wrappery dla Fortran-a. Przykładowe wywołanie

F77=ifort F90=ifort CC=icc **f2py3** -c -m ftest ftest.F90

Kod pliku ftest.F90

```
subroutine x(i,j,k)
  integer :: i
  integer :: j
  integer :: k

  !f2py intent(in) :: i
  !f2py intent(out) :: j,k

  j = i * 10
  write(*,*) j
  k = 10
end subroutine
```

Kod pliku test.py

```
import ftest
i = 100
j = ftest.x(i)
print(j)
```

Jako wyjście programu widzimy

```
1000
(1000, 10)
```