Programowanie w języku Fortran

dr inż. Maciej Woźniak [1]

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

Zapisanie do string-u “jedna szósta” i “jedna trzecia“

```
character (len=7) :: carr(2)
character(50) :: format1
format1 = ”(f7.4, /, f7.1)“
write (unit=carr, fmt=format1) 1.0/6.0, 1.0/3.0
carr(1) = ”0.1667“
carr(2) = ”0.3“
```

Jak zatem widać pierwszą gwiazdką w **write(*,*)** jest wyjście.
Drugą gwiazdką w **write(*,*)** jest formatowanie.

Otworzenie pliku “test.txt” pod 19-ką

```
open (unit(19), file=“text.txt”, position=“append”, &
  form=“formatted”, action=“read”, status=“old”)
...
open (unit(19), file=“text.txt”, position=“rewind”, &
  form=“unformatted”, action=“write”, status=“replace”)
...
write(unit=19,*) “test”
```

Opcje otwierania pliku

1. access
   - direct
   - sequential
   - stream
2. action
   - read
   - write
   - readwrite
3. form
   - formatted
   - unformatted

Opcje otwierania pliku

1. position
   - append
   - rewind
2. status
   - old
   - new
   - replace
   - scratch

Wywołanie C z poziomu Fortran-a.

```
void cprint (float * val){
  *val = 10.0;
}
```

```
use, intrinsic :: iso_c_binding
integer (kind=c_int) :: i
integer (kind=c_short) :: ishort
integer (kind=c_long) :: ilong
real (kind=c_float) :: creal
real (kind=c_double) :: cdouble
character (len=1, kind=c_char) :: cchar
```

Wywołanie C z poziomu Fortran-a.

```
interface
  subroutine cprint(val) bind(c)
    import :: c_float
    real (kind=c_float) :: val
  end subroutine cprint
end interface

creal = 100.0
call cprint(creal)
```

Wywołanie Fortrana z poziomu C.

```
subroutine X(i)
  ...
end subroutine

extern void x_(int *);
int main (int argc, char ** argv) {
  int i = 10;
  x_(&i);
return 0;
}
```

Rozszerzanie typów danych

```
type, public :: mtype
  real :: a, b, c
end type
```

```
type, public, extends (mtype) :: emtype
  integer :: d, e, f
end type
```

Rozszerzanie typów danych

Polimorfizm

class(mtype), **allocatable** :: mal
mal = mtype (1.0, 2.0, 3.0)
mal = emtype (1.0, 2.0, 3.0, 4, 5, 6)
allocate (mtype, **source**=emtype(1.0, 2.0, 3.0, 4, 5, 6))

Obiekty i klasy

```
type, public :: ctype
  real :: a
contains
  procedure, public :: ms
end type

procedure ms (a, b) result (c)
  class (ctype), intent(in) :: a
  real, intent(in) :: b
  real :: c
end subroutine

class(ctype) :: c
write(*,*) c%ms(1.0)
```

Coarrays

Coarrays są mechanizmem (pół)automatycznej równoległości w Fortran-ie 2008.
Proces jest tutaj nazywany obrazem (**image**).

- **num_images** - zwraca ilość procesów
- **this_image** - zwraca numer "tego" procesu

Całość działa według zasady **Single Program Multiple Data**.
Opcje kompilacji:

- **-fcoarray** - gfortran
- **-coarray** - ifort

Coarrays

Sposób deklaracji zmiennych dostępnych z innych obrazów

```
! kopia tablicy na każdym obrazie
real, dimension(10), codimension[*] :: c1
! dwuwymiarowa tablica z jednowymiarowym coarray
real, dimension (0:2, 4:8), codimension[0:*] :: c2
! pojedyncza zmienna
real, codimension[*] :: c3
```

Ważnym poleceniem jest **syncall**, które oczekuje aż wszystkie obrazy je wywołają.

Coarrays

Dostawanie się do zmiennych na innych obrazach

```
integer, codimension[*] :: c = -1
n = this_image() * 2 + 10
syncall()
if ( this_image() == 1 ) then
   write(*,*) n[1], n[2]
end if
```

Coarrays

W programowaniu równoległym istotne jest synchronizowanie dostępu do zmiennych

- **lock** - blokuje innym obrazom dostęp do zmiennej
- **unlock** - odblokowuje innym obrazom dostęp do zmiennej
- **critical**, **end critical** - oznaczenie sekcji krytycznej

```
integer, codimension[*] :: c
lock ( c[this_image()] )
...
unlock ( c[this_image()] )
```

Coarrays

Wielowymiarowe coarrays

integer, codimension [3,*] :: cint

mamy 4 obrazy
cint[1,1] -> obraz 1
cint[1,2] -> obraz 2
cint[1,3] -> obraz 3
cint[2,1] -> obraz 4

Sumarycznie wymiary i kowymiary macierzy nie mogą przekroczyć wartości 15.

Coarrays

Alokacja coarrays

```
integer, codimension [:], allocatable, dimension(:) :: cint
integer, codimension [:,:], allocatable, dimension(:,:) :: cint2
allocate(cint(10)[*])
allocate(cint2(5,10)[3,*])
```

Coarrays

Ograniczenia techniczne:

- jeśli przekazywany parametr subroutiny ma zdefiniowany kowymiar, to faktycznie przekazany też musi mieć
- wynik działania funkcji nie może mieć kowymiaru
- parametr subroutiny, który ma kowymiar nie może mieć **intent**(out)
- składowa obiektu z kowymiarem nie może posiadać swojego kowymiaru

Co pozostało

- wskaźniki na funkcje
- systemy dokumentacji
- python + Fortran
- programowanie funkcyjne
- obsługa stringów