

# C2115

# Praktický úvod do superpočítání

III. lekce

Petr Kulhánek, Jakub Štěpán

[kulhanek@chemi.muni.cz](mailto:kulhanek@chemi.muni.cz)

Národní centrum pro výzkum biomolekul, Přírodovědecká fakulta,  
Masarykova univerzita, Kotlářská 2, CZ-61137 Brno

## ➤ Úvod

historie jazyka Fortran, Hello world!, překladače, překládáme, volby překladače

## ➤ Syntaxe

program, rozdíly oproti F77, proměnné, řídicí struktury, I/O, pole, funkce, procedury

## ➤ Cvičení

jednoduché programy, násobení matic, výpočet určitého integrálu

## ➤ Literatura

# Úvod

---

# Historie

**Fortran** (zkratka slov FORmula a TRANslator) je v informatice imperativní programovací jazyk, který v 50. letech 20. století navrhla firma IBM pro **vědecké výpočty a numerické aplikace**.

Zdroj: wikipedia

## Verze jazyka:

Fortran 77

**Fortran 90**

**Fortran 95**

**Fortran 2003**

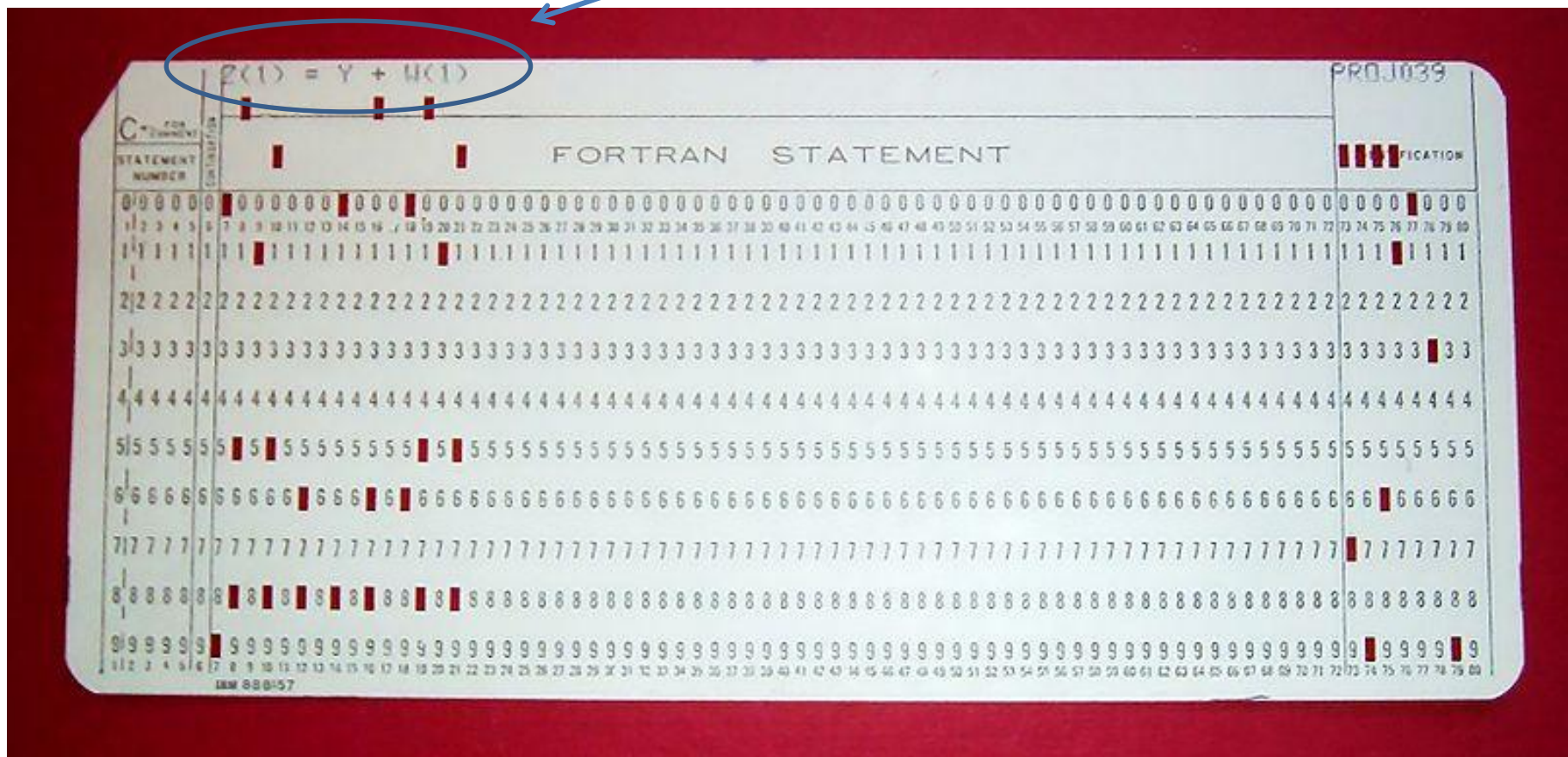
**Fortran 2008**

V tomto jazyce nebo pro tento jazyk je napsána **celá řada knihoven**. Kompilátory jsou schopny vytvářet **velmi optimalizovaný kód**.

**Standardní matematické knihovny:** BLAS, LAPACK a další na <http://www.netlib.org>

# Historie

jeden zdrojový řádek



Zdroj: wikipedia

# Hello world!

hello.f90

```
program Hello  
  
write(*,*) 'Hello world!'  
  
end program
```

**Kompilace:**

```
$ gfortran hello.f90 -o hello
```

**Spuštění:**

```
$ ./hello
```

**Kompilace do assembleru:**

```
$ gfortran hello.f90 -S  hello.s
```

# Cvičení LIII.1

1. Vytvořte soubor hello.f90. Zkompilujte jej překladačem gfortran. Ověřte funkci vytvořeného programu.

# Překladače

## GNU GCC

Překladač: **gfortran**

Typ licence: GNU GPL (volně dostupný)

URL: <http://gcc.gnu.org/wiki/GFortran>

## Intel® Composer XE

Překladač: **ifort**

Typ licence: a) komerční (dostupný v MetaCentru, meta moduly: intelcdk)  
b) zdarma k osobnímu použití proti registraci (linux)

URL: <http://software.intel.com/en-us/articles/intel-composer-xe/>

## The Portland Group

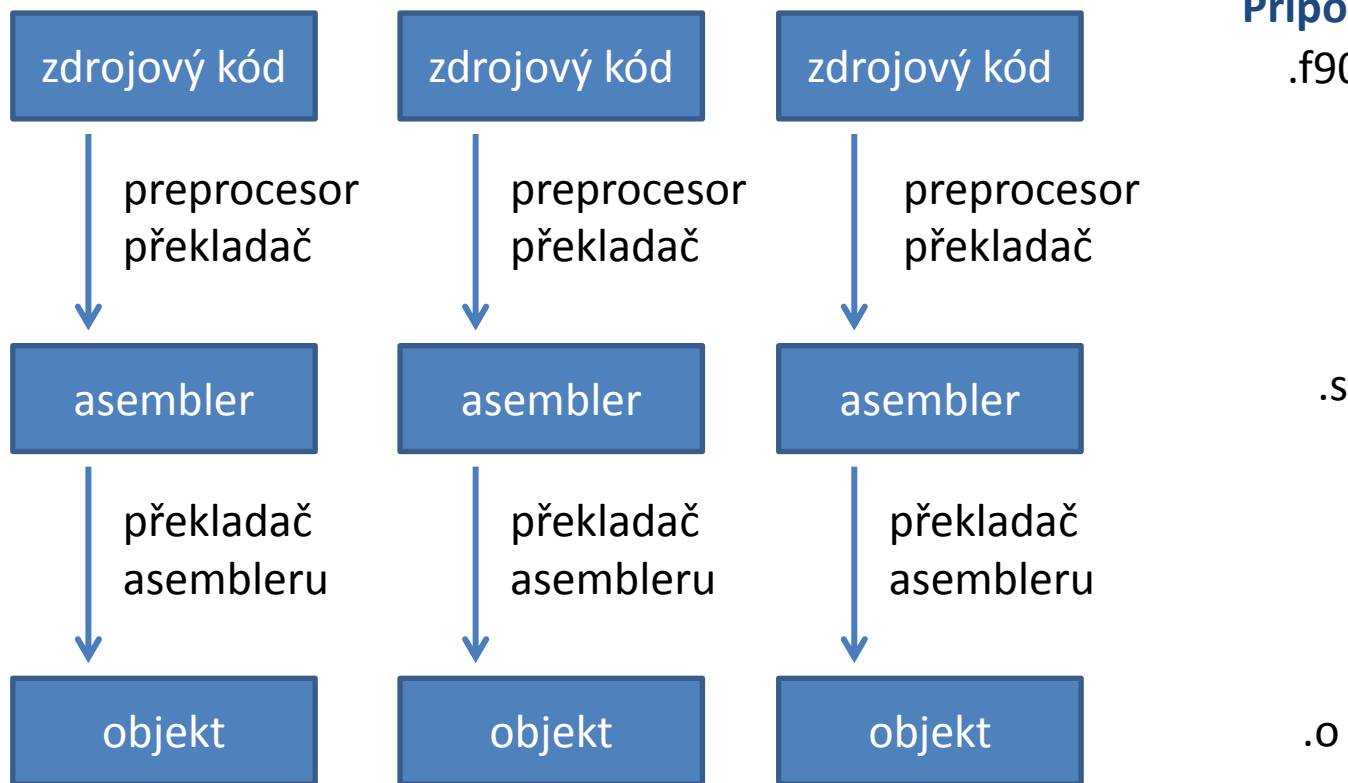
Překladač: **pgf90, pgf77**

Typ licence: komerční (dostupný v MetaCentru, meta moduly: pgicdk)

URL: <http://www.pgroup.com/>



# Překládáme ...

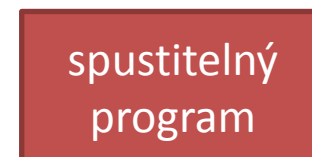
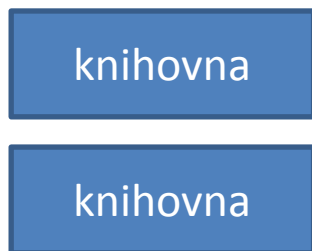


Přípona:  
.f90

.s

.o

Přípona: .so, .a



# Užitečné volby překladače

## Volby překladače:

- o název výsledného programu
- c přeloží zdrojový kód do objektového kódu
- S přeloží zdrojový kód do assembleru
- Ox úroveň optimalizace výsledného programu, kde x=0 (žádná), 1, 2, 3 (nejvyšší)
- g vloží dodatečné informace a kód pro ladění běhu programu (zpomaluje běh programu)
- lname připojení (linkování) knihovny *name* k výslednému programu
- Lcesta cesta ke knihovnám, které nejsou ve standardních cestách

## Volby překladače (ifort):

- trace all kontroluje meze polí, použití neinicializovaných proměnných, atd.

# Programy napsané ve Fortranu

## Gaussian

<http://www.gaussian.com/>

Komerční program určený pro kvantově chemické výpočty.

## AMBER

<http://www.ambermd.org/>

Akademický software určený k molekulárním simulacím za použití molekulové mechaniky a hybridních QM/MM metod. Ve Fortranu jsou napsány programy **sander** a **pmemd**.

## CPMD

<http://www.cpmd.org/>

Akademický software určený pro molekulární simulace za použití metod funkcionálu hustoty.

**Další software:** Turbomole, DALTON, CP2K, ABINIT a další ...

[http://en.wikipedia.org/wiki/List\\_of\\_quantum\\_chemistry\\_and\\_solid\\_state\\_physics\\_software](http://en.wikipedia.org/wiki/List_of_quantum_chemistry_and_solid_state_physics_software)

# Syntaxe

---

# F77 dialekt

- fixní formát
- sloupec 1, pokud začíná písmenem C jedná se o komentář
- sloupec 1-6 je vyhrazen pro návěští (pro I/O formáty, cykly)
- sloupec 6, pokud obsahuje znak \* jedná se o pokračování předchozího řádku
- sloupec 7-72 obsahuje řádek programu

```
12345678901234567890123456789001234567891234567890123456789012345678900123456789
C toto je komentar
  implicit none
  real      f
  integer   a, b
C -----
C secti cisla a a b
  a = a + b
C dlouhy radek
  f = a*10.0 + 11.2*b
  *+ (a+b)**2
100  format(I10)
     write(*,100) a
```

# Zdrojové soubory

- Fortran 90 a výše používá volnou syntaxi (příkazy již není nutné zarovnávat do sloupců jako tomu bylo u Fortran 77).
- Povolené zakončení názvů zdrojových souborů: .fpp, **.f90**, .f95, .f03, .f08
- Fortran není case-sensitive (tj. nerozlišuje se velikost písma)
- K odsazování není vhodné používat tabulátor.
- Komentáře mohou začínat kdekoliv, k uvození komentáře se používá vykřičník !.
- Maximální délka řádku je omezena (typicky 132 znaků). Pro zápis delších výrazů se používá znak ampersand &.

```
implicit none
real          :: f
integer       :: A, B
! -----
! secti cislo A a B
A = A + B
f = A*10.0 + 11.2*B &
    + (A+B)**2      ! dlouhy radek
```

# Preprocesor

- Zdrojový soubor může obsahovat direktivy CPP preprocesoru (používaného jazyky C a C++)

**#include <soubor>**

**#include "soubor"**

**#ifdef**

**#ifndef**

**#if**

**#else**

**#endif**

**#define**

a další ...

- Zpracování souboru preprocesorem lze vynutit volbou kompilátoru, popř. změnou zakončení souboru na: .fpp, .FPP, F90, .F95, .F03, .F08

<http://gcc.gnu.org/onlinedocs/gfortran/Preprocessing-Options.html>

# Sekce Program

```
program Hello  
! definice promennych  
  
! vlastni program  
write(* ,*) ' Hello world! '  
  
! Konec programu  
end program
```



směr vykonávání programu

Program může být předčasně ukončen příkazem **stop**.



# Proměnné

```
implicit none
```

```
logical      :: f
```

```
integer     :: a, g
```

```
real        :: c, d
```

```
double precision :: e
```

```
character(len=30) :: s
```

**vypne automatickou deklaraci proměnných**

reálné číslo v jednoduché přesnosti

reálné číslo v dvojnásobné přesnosti

řetězec (text)

maximální délka řetězce ve znacích

## Alternativní zápisy:

```
real(4)      :: c, d
```

```
real(8)     :: e
```

Proměnné definujeme na začátku programu, funkce nebo procedury.

# Proměnné

```
implicit none
logical          :: f
! -----
f = .TRUE.
write(*,*) f
f = .FALSE.
write(*,*) f
```

```
implicit none
real            :: a,b
! -----
a = 1.0
b = 2.0
b = a + b
write(*,*) a, b
```

```
implicit none
character(len=30) :: s
! -----
s = 'pokusny text'
write(*,*) trim(f)
```

**Proměnné vždy inicializujeme**  
(tj. přiřadíme jim výchozí hodnotu).

funkce **trim** ořízne řetězec zprava (odstraní prázdné znaky)

# Proměnné

```
implicit none
real      :: a = 1.0
real      :: b
! -----
b = 2.0
b = a + b
write(*,*) a, b
```

**NIKDY** neinicializujeme  
proměnnou během její deklarace.

povolená konstrukce, která se překládá jako

```
real, save      :: a = 1.0
```

obdoba klíčového slova "**static**" z jazyka C a C++

# Matematické operace

## Operátory:

|    |          |
|----|----------|
| +  | sčítání  |
| -  | odčítání |
| *  | násobení |
| /  | dělení   |
| ** | umocnění |

## Bez přímé podpory:

**MOD(n,m)** modulo ( $n \% m$  z jazyka C)

```
real          :: a, b, c
```

```
! -----
```

```
a = 1.0
```

```
b = 2.0
```

```
c = 4.0
```

```
b = a + b
```

```
b = a * b / c
```

```
c = a ** 2 + b ** 2
```

# Cykly I

```
do promenna = pocatecni_hodnota, koncova_hodnota [, krok]
    prikaz1
    prikaz2
    ...
end do
```

Proměnná může být pouze **celé číslo (integer)**.

```
integer          :: i
!-----
do i = 1, 10
    write(*,*) i
end do
```

Vypíše čísla: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

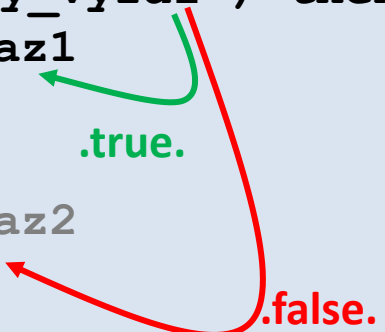
```
integer          :: i
!-----
do i = 1, 10, 2
    write(*,*) i
end do
```

Vypíše čísla: 1, 3, 5, 7, 9

Běh cyklů může být řízen příkazy **cycle** (obdoba continue z jazyka C) a **exit** (obdoba break).

# Podmínky

```
if ( logicky_vyraz ) then
    prikaz1
    ...
else
    prikaz2
    ...
end if
```



```
integer          :: i = 7
!-----
if( i .gt. 5 ) then
    write(*,*) 'i je vetsi nez 5'
end if
```

## Logické operátory:

**.and.** logické ano  
**.or.** logické nebo  
**.not.** negace

## Porovnávací operátory (čísla):

**.eq.** rovná se  
**.ne.** nerovná se  
**.lt.** menší než  
**.le.** menší než nebo rovno  
**.gt.** větší než  
**.ge.** větší než nebo rovno

## Porovnávací operátory (logical):

**.eqv.** ekvivalence  
**.neqv.** neekvivalence

# Cykly II

```
do while ( logicky_vyraz )
    prikaz1
    prikaz2
    ...
end do
```

cyklus probíhá dokud  
**logicky\_vyraz** vraci **.true.**

```
double precision      :: a
!-----
a = 0.0
do while ( a .le. 5 )
    write(*,*) a
    a = a + 0.1
end do
```

Vypíše čísla od 0 do 5 s krokem 0.1

Běh cyklů může být řízen příkazy **cycle** (obdoba continue z jazyka C) a **exit** (obdoba break).

# Funkce a procedury

**Funkce** je část programu, kterou je možné **opakovaně** volat z různých míst kódu.

**Procedura** je podobná funkci, ale na rozdíl od funkce **nevrací hodnotu**. Vhodným použitím funkcí a procedur se zvyšuje čitelnost programu a snižuje se duplicitní kód.

```
program Hello
! definice promenných
...
! vlastní program
! volání funkce nebo procedury
...
! konec programu
...
contains

! definice funkce nebo procedur

end program
```

Funkce a procedury lze volat jak z vlastního programu, tak i samotných funkcí a procedur.

**Argumenty** funkcí a procedur se **předávají odkazem**.



# Definice funkce

```
function moje_funkce(a,b,c) result(x)
implicit none
double precision :: a, b, c ! argumenty (parametry) funkce
double precision :: x      ! vysledek funkce
!-----
integer          :: j      ! lokalni promenna
!-----
! vlastni telo funkce
x = a + b + c
end function moje_funkce
```

## Alternativní zápis:

```
double precision function moje_funkce(a,b,c)
...
moje_funkce = a + b + c
end function moje_funkce
```

# Definice procedury

```
subroutine moje_procedura(a,b,c)
implicit none
double precision :: a, b, c ! argumenty (parametry) procedury
!-----
integer          :: j      ! lokalni promenna
!-----
! vlastni telo procedury
a = a + b + c
end subroutine moje_procedura
```

Přístupové vlastnosti argumentů funkcí a procedur lze měnit pomocí klíčového slova **intent**. Defaultní přístupovou vlastností je **intent(inout)**.

|  |      |                                       |
|--|------|---------------------------------------|
| double precision, intent( <b>in</b> )    | :: a | ! argument lze pouze číst             |
| double precision, intent( <b>out</b> )   | :: b | ! do argumentu lze pouze zapisovat    |
| double precision, intent( <b>inout</b> ) | :: c | ! s argumentem lze pracovat libovolně |

# Volání funkcí a procedur

## Volání funkcí:

```
double precision :: a
double precision :: d
!-----
a = 5.0
d = moje_funkce_2(a)
write(*,*) d
```

```
double precision :: a
double precision :: d
!-----
a = 5.0
moje_funkce_2(a)
```

## Volání procedur:

```
double precision :: a
double precision :: d
!-----
a = 5.0
d = 2.0
call moje_procedura_3(a,d)
```

Výsledek funkce se **musí použít**.

# Předávání argumentů odkazem

```
double precision :: a  
double precision :: d
```

```
!-----
```

```
a = 5.0
```

```
d = 2.0
```

```
write(*,*) d
```

```
call moje_procedura_3(a,d)
```

```
write(*,*) d
```

2

?

```
subroutine moje_procedura_3(a,b)
```

```
implicit none
```

```
double precision :: a, b    ! argumenty (parametry)
```

```
!-----
```

```
! vlastní telo procedury
```

```
b = a + b
```

```
end subroutine moje_procedura_3
```

# Předávání argumentů odkazem

```
double precision :: a  
double precision :: d
```

```
!-----
```

```
a = 5.0
```

```
d = 2.0
```

```
write(*,*) d
```

```
call moje_procedura_3(a,d)
```

```
write(*,*) d
```

2

7

V jazyce C by byla  
hodnota rovna 2.

```
subroutine moje_procedura_3(a,b)
```

```
implicit none
```

```
double precision :: a, b    ! argumenty (parametry)
```

```
!-----
```

```
! vlastní telo procedury
```

```
b = a + b
```

```
end subroutine moje_procedura_3
```

# Některé standardní funkce a procedury

## Matematické funkce:

|                       |                             |
|-----------------------|-----------------------------|
| <code>sin(x)</code>   |                             |
| <code>cos(x)</code>   |                             |
| <code>sqrt(x)</code>  | druhá odmocnina             |
| <code>exp(x)</code>   |                             |
| <code>log(x)</code>   | <b>přirozený</b> logaritmus |
| <code>log10(x)</code> | dekadický logaritmus        |

## Náhodné čísla:

|                                       |  |
|---------------------------------------|--|
| <code>call random_seed()</code>       | inicializuje generátor náhodných čísel                                   |
| <code>call rand_number(number)</code> | nastaví proměnou <b>number</b> na náhodné číslo v intervalu $<0.0;1.0$ ) |

## Měření času:

|                                  |  |
|----------------------------------|--|
| <code>call cpu_time(time)</code> | nastaví hodnotu proměnné <b>time</b> na čas běhu programu v sekundách (s mikrosekundovým rozlišením) |
|----------------------------------|--|

# Pole

## Statically defined arrays:

```
double precision :: a(10)
double precision :: d(14,13)
```

Jednorozměrné pole o velikosti 10 prvků.

Dvourozměrné pole o velikosti 14x13 prvků.  
(14 řádků a 13 sloupců)

## Dynamically declared arrays:

```
double precision, allocatable :: a(:)
double precision, allocatable :: d(:, :)
! -----
! alocace pameti pro pole
allocate(a(10000), d(200,300))
! pouziti pole
! uvolneni pameti
deallocate(a,d)
```

Jednorozměrné pole-

Dvourozměrné pole.

Rozměry polí mohou být  
definovány i pomocí  
celočíslných proměnných.

# Práce s polem

```
double precision :: a(10)
double precision :: d(14,13)
integer          :: i
!-----

a(:) = 0.0  ! lze zapsat i jako a = 0.0
do i=1, 10
    write(*,*) i, ' - ty prvek pole je ', a(i)
end do

a = d(:,1) ! zapiše první sloupec z
           ! matice d do vektoru a

a(5) = 2.3456
d(1,5) = 1.23
write(*,*) d(1,5)
```

**Prvky pole se indexují od jedné.\***

Velikost pole lze zjistit funkcí **size**.

\* rozsahy indexů pro jednotlivé rozměry lze však měnit

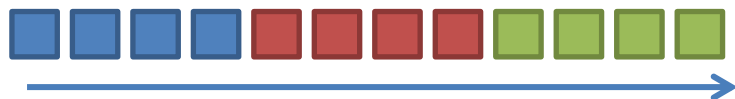
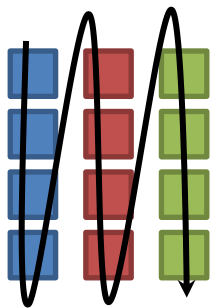


# Pole – paměťový model

Fortran

$a(i,j)$

Prvky jsou za sebou ve sloupcích  
(column based).

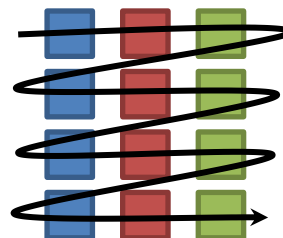


uspořádání prvků matice v paměti

C/C++

$A[i][j]$

Prvky jsou za sebou v řádcích  
(row based).



Pokud voláme funkce z knihoven BLAS či LAPACK nutno počítat s rozdílným indexováním vícerozměrných polí.

# Pole – paměťový model

## Fortran

```
double precision :: d(10,10)
double precision :: sum
integer          :: i,j
!-----

sum = 0.0d0
do i=1, 10
  do j=1,10
    sum = sum + d(j,i)
  end do
end do
```

index se mění nejrychleji pro řádky

## C/C++

```
double* d[];
double sum;
//-----

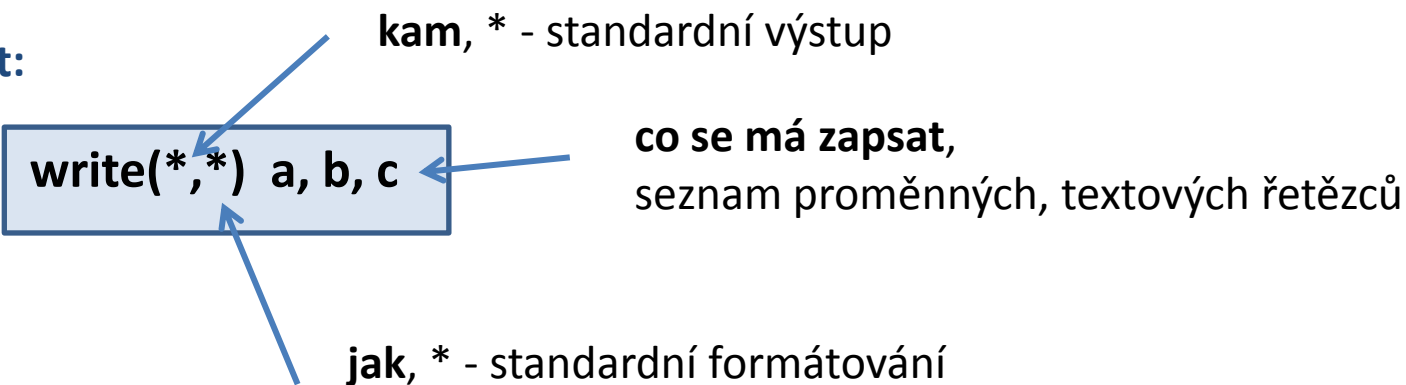
sum = 0.0;
for(int i=0; i < 10; i++){
  for(int j=0; j < 10; j++){
    sum += d[i][j];
  }
}
```

index se mění nejrychleji pro sloupce

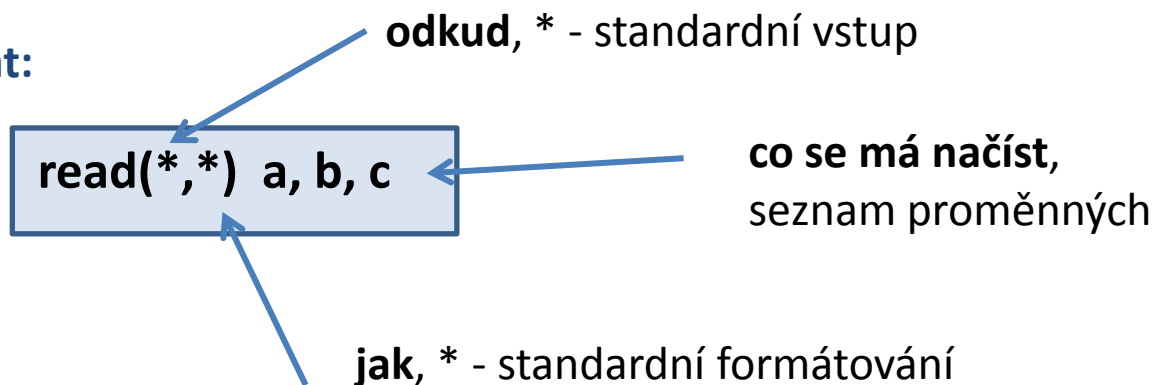
**Poznámka:** uvedené uspořádání nemá vliv na funkci, ale na rychlost vykonávání

# I/O operace

Zápis dat:



Čtení dat:



Soubory se otevírají příkazem **open**. Zavírají příkazem **close**.

# I/O operace - formátování

## Formátovaný výstup:

```
write(*,10) a, b, c
```

```
10 format('Hodnota a=',F10.6,' hodnota b=',F10.6, ' hodnota c=',F10.6)
```

- formát může být uveden před i za příkazem write či read
- formátovací typy:
  - F – reálné číslo ve fixním formátu
  - E – reálné číslo ve vědeckém formátu
  - I – celé číslo
  - A - řetězec

## Zápis dat bez vypsaní znaku konce řádku:

```
write(*,10,ADVANCE='NO') a, b, c
```

 musí být uveden formát

# Další vlastnosti jazyka

1. podpora pointerů
2. struktury
3. objektově orientované programování

# Cvičení

---

Všechny data a zdrojové kódy vyřešených cvičení dáváme do samostatných adresářů. Budeme je využívat v dalších částech kurzu.

# Cvičení LIII.2

1. Napište program, který vypíše do terminálu deset znaku 'A', každý na samostatný řádek.
2. Napište program, který vypíše do terminálu deset znaku 'A' vedle sebe.
3. Napište program, který vypíše do terminálu písmena A uspořádané do pravoúhlého trojúhelníka.
4. Napište program, který vypíše do terminálu písmena A uspořádané do čtverce.

# Cvičení LIII.3

1. Vytvořte program, který dynamicky alokuje dvojrozměrné pole **A** o velikosti  $n \times n$ . Prvky pole inicializujte náhodnými čísly z intervalu  $\langle -10 ; 20 \rangle$ . Pole vytiskněte na obrazovku.
2. Vytvořte dvě dvourozměrné pole (matice) **A** a **B** o rozměrech  $n \times n$ . Pole inicializujte náhodnými čísly stejným způsobem jako v předchozím cvičení. Napište program pro násobení matic **A** a **B**. Výsledek uložte do matice **C**.
3. Kolik operací v plovoucí čárce program vykoná při násobení matic? Změřte dobu, za kterou program uskuteční násobení matic (do času nezapočítávejte vytvoření a inicializaci matic A a B). Z počtu operací a doby potřebné k násobení stanovte přibližný výkon procesoru v MFLOPS.
4. Změřte výkon procesoru pro různé velikosti matic **A** a **B**. Naměřené hodnoty pro  $n$  v intervalu 10 až 1000 vynesete do grafu a vyhodnoťte.



# Cvičení LIII.4

1. Napište program, který vypočte určitý integrál uvedený níže. K integraci použijte lichoběžníkovou metodu.

$$I = \int_0^1 \frac{4}{1+x^2} dx$$

2. Čemu se integrál rovná? Zjištění zdůvodněte.

# Domací úkol

<http://summerofhpc.prace-ri.eu/>

1. Vyřešte první úkol **code testu**.

# Literatura

---

- <http://www.root.cz/serialy/fortran-pro-vsechny/>
- <http://gcc.gnu.org/onlinedocs/gfortran/>
- Dokumentace ke kompilátoru ifort
- Clerman, N. S. Modern Fortran: style and usage; Cambridge University Press: New York, 2012.