Programowanie

Krótki wstęp do Fortrana

Janusz Szwabiński

Plan wykładu

- Wprowadzenie
- Instalacja
- Pierwszy program w Fortranie
- Typy danych
- Prosty program w Fortranie
- Stałe
- Wyrażenia warunkowe
- Pętle
- Procedury i funkcje
- Tablice
- Łańcuchy znaków
- Wskaźniki
- Struktury danych
- Moduły

Literatura:

1. Roger Kaufman, *A FORTRAN Coloring Book*
2. Stephen J. Chapman, *Fortran for Scientists and Engineers*
3. https://www.tutorialspoint.com/fortran/index.htm (https://www.tutorialspoint.com/fortran/index.htm)

Wprowadzenie

- Fortran (od wersji 90 do aktualnej), a dawniej FORTRAN (do wersji 77 włącznie)
- od ang. *FORmula TRANslator*
- język programowania pierwotnie zaprojektowany do zapisu programów obliczeniowych
- wcześniej był językiem proceduralnym
- obecnie umożliwia programowanie strukturalne, obiektowe (Fortran 90/95), modularne i równoległe (Fortran 2008)
- zastosowania:
  - obliczenia naukowo-inżynierskie
  - obliczenia numeryczne
  - symulacje komputerowe
- mimo początkowo dość ograniczonych możliwości, dzięki łatwości opanowania Fortran stał się najpopularniejszym językiem do obliczeń numerycznych
- współcześnie często pomijany w IT i dydaktyce, jednak:
  - znacząca liczba projektów naukowych jest nadal liczona w Fortranie
  - został on zauważony w kulturze masowej

In [1]:

```
from IPython.display import HTML
```

In [2]:

```
%%HTML
<video width="320" height="240" controls>
  <source src="Greatest Programming Language.mp4" type="video/mp4">
</video>
```

0:00 / 0:19

(Simpsonowie, Sezon 26, Odcinek 10, źródło: YouTube)

Historia

- pierwszy kompilator Fortrana stworzył zespół Johna Backusa, który w latach 1954-1957 pracował dla IBM
  - pierwszy w historii kompilator języka wysokiego poziomu
  - starannie zoptymalizowany, aby szybkość programów była porównywalna z szybkością programów napisanych w asemblerze
- Fortran IV - pierwsza standaryzacja nastąpiła (1960)
- Fortran 66 - kolejny standard
  - dość ubogi
  - implementacje musiały zawierać wiele rozszerzeń
- Fortran 77 - standard opracowany przez American National Standard Institute (ANSI)
  - od roku 1980 standard międzynarodowy
  - struktura przystosowana do używanych wówczas powszechnie kart perforowanych
  - elementy programowania strukturalnego
  - pętle wstecz
  - bloki `IF...THEN...ELSE...ENDIF`
  - dane tekstowe (typ `CHARACTER`)
  - apostrofy jako ograniczniki pola tekstowego
  - nadal w powszechnym użyciu

In [7]:

```
%%HTML
<video width="320" height="240" controls>
  <source src="Punching Data.mp4" type="video/mp4">
</video>
```

0:00 / 4:14

- Fortran 90
  - zmieniona składnia, dostosowana do współczesnych języków programowania
  - swobodny format
  - operacje na tablicach
  - instrukcje `CASE` i `DO WHILE`
  - możliwość definiowania własnych operatorów
  - moduły jako pakiety zawierające kod i zmienne
- Fortran 95
  - zmiany w stosunku do poprzedniej wersji są niewielkie
- Fortran 2008
  - programowanie obiektowe
  - wkaźniki procedur
  - rozszerzenie definicji typów
  - obsługa wyjątków
  - współpraca z modułami napisanymi w C
  - obsługa mieðzynarodowych standardów danych
  - dostęp do argumentów linii poleceń, zmiennych środowiskowych i komunikatów procesora o błędach.

Instalacja i środowisko

Kompilatory

Linux/Unix

- `gfortran` - kompilator z rodziny GCC (GNU Compilers Colletion)
- `g95`

Windows

- `Silverfrost FTN95`
  - darmowy w wersji *Personal Edition*
- `gfortran`
  - część projektu MinGW
- `g95`

MacOS

- `gfortran`
- `g95`

Edytory

- gedit, emacs, vim, jed, geany, kate (Linux/Unix)
- Notepad++, Notetab light (Windows)

IDE

- Code::Blocks (różne systemy operacyjne)
- Simply Fortran (Windows)
- Photran (część środowiska Eclipse)

Instalacja pod Ubuntu

```
sudo apt-get install gfortran
```

Pierwszy program w Fortranie

In [3]:

```
%%writefile hello.f90
program hello

    print *, "Hello, World!"

end program hello
```

```
Writing hello.f90
```

In [4]:

```
!gfortran hello.f90 -o hello
```

In [6]:

```
!./hello
```

```
 Hello, World!
```

Typy danych ¶

Podstawowe typy danych to:

Typ	Oznaczenie
Całkowity	`integer`
Zmiennoprzecinkowy	`real`
j.w. (podwójna precyzja)	`double precision`
Zespolony	`complex`
Logiczny	`logical`
Znakowy	`character`

- tradycyjnie w Fortranie mamy dwa typy zmiennoprzecinkowe, `real` i `double precision`
- w Fortranie 90/95 mamy więcej kontroli nad precyzją dzięki specyfikatorowi `KIND`
  - programista może określić długość reprezentacji liczby w bajtach
- starsze wersje Fortrana dopuszczały tzw. typowanie niejawne (*implicit typing*)
  - jeśli zmienna nie była zadeklarowana, pierwsza litera jej nazwy określała jej typ
  - zmienne o nazwach zaczynających się literą z przedziału `i`-n były typu całkowitego
  - pozostałe były liczbami zmiennoprzecinkowymi
  - typowanie niejawne można wyłączyć poleceniem `implicit none`

In [8]:

```
%%writefile ints.f90
program ints
implicit none

   !liczba całkowita 2-bajtowa
   integer(kind = 2) :: shortval

   !liczba całkowita 4-bajtowa
   integer(kind = 4) :: longval

   !liczba całkowita 8-bajtowa
   integer(kind = 8) :: verylongval

   !liczba całkowita 16-bajtowa
   integer(kind = 16) :: veryverylongval

   !domyślna liczba całkowita
   integer :: defval

   print *, huge(shortval)
   print *, huge(longval)
   print *, huge(verylongval)
   print *, huge(veryverylongval)
   print *, huge(defval)

end program ints
```

```
Writing ints.f90
```

In [9]:

```
!gfortran ints.f90 -o ints
!./ints
```

```
  32767
  2147483647
  9223372036854775807
 170141183460469231731687303715884105727
  2147483647
```

In [16]:

```
%%writefile floats.f90
program floats
implicit none

   !liczba zmiennoprzecinkowa 4-bajtowa
   real(kind = 4) :: longval

   !liczba zmiennoprzecinkowa 8-bajtowa
   real(kind = 8) :: verylongval

   !liczba zmiennoprzecinkowa 16-bajtowa
   real(kind = 16) :: veryverylongval

   !domyślna liczba zmiennoprzecinkowa
   real :: defval

   print *, huge(longval)
   print *, huge(verylongval)
   print *, huge(veryverylongval)
   print *, huge(defval)

end program floats
```

```
Overwriting floats.f90
```

In [17]:

```
!gfortran floats.f90 -o floats
!./floats
```

```
   3.40282347E+38
   1.7976931348623157E+308
   1.189731495357231765085759326628007 02E+4932
   3.40282347E+38
```

In [46]:

```
%%writefile complex.f90
program compar
implicit none

   complex, parameter :: i = (0, 1)   ! sqrt(-1)
   complex :: x, y, z

   x = (7, 8)
   y = (5, -7)
   write(*,*) i * x * y

   z = x + y
   print *, "z = x + y = ", z

   z = x - y
   print *, "z = x - y = ", z

   z = x * y
   print *, "z = x * y = ", z

   z = x / y
   print *, "z = x / y = ", z

end program compar
```

```
Writing complex.f90
```

In [47]:

```
!gfortran complex.f90 -o complex
!./complex
```

```
 (  9.00000000    ,  91.0000000    )
 z = x + y =  (  12.0000000    ,  1.00000000    )
 z = x - y =  (  2.00000000    ,  15.0000000    )
 z = x * y =  (  91.0000000    , -9.00000000    )
 z = x / y =  (-0.283783793    ,  1.20270276    )
```

Prosty program w Fortranie

In [18]:

```
%%writefile suma.f90
program suma
   implicit none
   !deklaracje zmiennych
   real :: a, b, result

   !wykonywalna część programu
   a = 12.0
   b = 15.0
   result = a + b
   print *, 'Suma liczb wynosi ', result

end program suma
```

```
Writing suma.f90
```

In [19]:

```
!gfortran suma.f90 -o suma
!./suma
```

```
 Suma liczb wynosi    27.0000000
```

Stałe

In [20]:

```
%%writefile parameters.f90
program params

   ! przyspieszenie ziemskie (stała)
   real, parameter :: g = 9.81

   ! deklaracje zmiennych
   real :: s ! droga
   real :: t ! czas
   real :: u ! prędkość początkowa

   t = 5.0
   u = 50

   ! oblicz drogę
   s = u * t - g * (t**2) / 2

   ! wypisz wyni na ekran
   print *, "Czas = ", t
   print *, 'Droga = ',s

end program params
```

```
Writing parameters.f90
```

In [21]:

```
!gfortran parameters.f90 -o params
!./params
```

```
 Czas =    5.00000000
 Droga =    127.374992
```

In [22]:

```
%%writefile parameters2.f90
program params2

   ! przyspieszenie ziemskie (stała)
   real, parameter :: g = 9.81

   ! deklaracje zmiennych
   real :: s ! droga
   real :: t ! czas
   real :: u ! prędkość początkowa

   t = 5.0
   u = 50

   g = 10.0 !ta operacja powinna być niedopuszczalna

   ! oblicz drogę
   s = u * t - g * (t**2) / 2

   ! wypisz wyni na ekran
   print *, "Czas = ", t
   print *, 'Droga = ',s

end program params2
```

```
Writing parameters2.f90
```

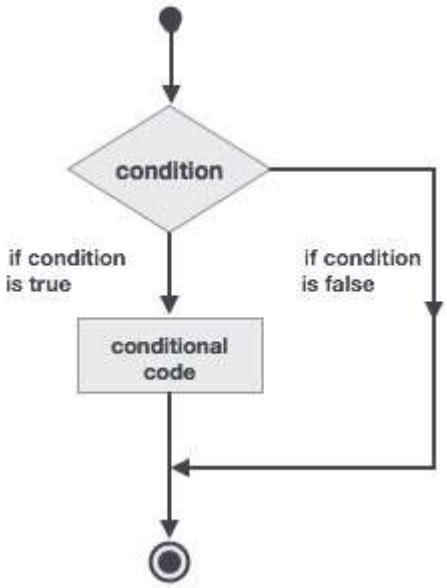
In [23]:

```
!gfortran parameters2.f90 -o params2
```

```
parameters2.f90:14:3:

    g = 10.0
   1
Error: Named constant ‘g’ in variable definition context (assignmen
t) at (1)
```

Wyrażenia warunkowe



Wyrażenie `if`

In [34]:

```
%%writefile ifstat.f90
program ifstat
implicit none
   ! deklaracja zmiennych
   integer :: a = 10

   ! wyrażenie warunkowe
   if (a < 20 ) then
      print*, "a jest mniejsze od 20"
   end if

   print*, "wartość a wynosi ", a
 end program ifstat
```

```
Writing ifstat.f90
```

In [35]:

```
!gfortran ifstat.f90 -o ifstat
!./ifstat
```

```
 a jest mniejsze od 20
 wartość a wynosi           10
```

In [36]:

```
%%writefile ifstat2.f90
program ifstat2
implicit none

   ! deklaracja zmiennych
   integer :: a = 100

   ! sprawdź wartość
   if( a == 10 ) then
      print*, "Wartość a wynosi 10"
   else if( a == 20 ) then
      print*, "Wartość a wynosi 20"
   else if( a == 30 ) then
      print*, "Wartość a wynosi 30"
   else
      print*, "Żadna z wartości nie pasuje"
   end if

   print*, "Wartość a wynosi ", a

end program ifstat2
```

```
Writing ifstat2.f90
```

In [37]:

```
!gfortran ifstat2.f90 -o ifstat2
!./ifstat2
```

```
 Żadna z wartości nie pasuje
 Wartość a wynosi          100
```

Konstrukcja `select...case`

In [40]:

```
%%writefile ocena.f90
program ocena
implicit none

   ! deklaracja zmiennych
   character :: grade = 'E'

   !właściwa konstrukcja
   select case (grade)
      case ('A')
          print*, "Wyśmienicie!"
      case ('B')
          print*, "Bardzo dobrze!"
      case ('C')
         print*, "Dobrze"

      case ('D')
         print*, "Zdałeś"

      case ('E')
         print*, "Do zobaczenia w przyszłym roku"

      case default
         print*, "Ocena nieważna"
   end select

   print*, "Twoja ocena: ", grade

end program ocena
```

```
Overwriting ocena.f90
```

In [41]:

```
!gfortran ocena.f90 -o ocena
!./ocena
```

```
 Do zobaczenia w przyszłym roku
 Twoja ocena: E
```

In [44]:

```
%%writefile ocena2.f90
program ocena2
implicit none

   integer :: marks = 78

   select case (marks)

      case (91:100)
         print*, "Wyśmienicie!"

      case (81:90)
         print*, "Bardzo dobrze!"

      case (71:80)
         print*, "Dobrze!"

      case (61:70)
         print*, "Nieźle!"

      case (41:60)
         print*, "Zdałeś!"

      case (:40)
         print*, "Dramat!"

      case default
         print*, "Ocena nieważna"

   end select
   print*, "Twoja ocena: ", marks

end program ocena2
```

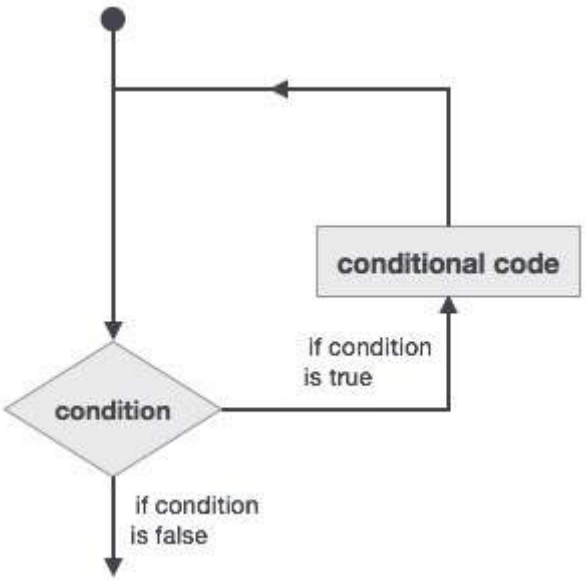
```
Writing ocena2.f90
```

In [45]:

```
!gfortran ocena2.f90 -o ocena2
!./ocena2
```

```
 Dobrze!
 Twoja ocena:           78
```

Pętle



Pętle do

In [24]:

```
%%writefile silnia.f90
program silnia
implicit none

   ! deklaracja zmiennych
   integer :: nfact = 1
   integer :: n

   ! wylicz kolejne wartości silni
   do n = 1, 10   !pętla po n od 1 do 10 (włącznie)
      nfact = nfact * n
      ! wypisz na ekran
      print*,  n, " ", nfact
   end do

end program silnia
```

```
Writing silnia.f90
```

In [25]:

```
!gfortran silnia.f90 -o silnia
!./silnia
```

```
           1             1
           2             2
           3             6
           4            24
           5           120
           6           720
           7          5040
           8         40320
           9        362880
          10       3628800
```

Pętle do `while`

In [28]:

```
%%writefile silnia2.f90
program silnia2
implicit none

   ! deklracacja zmiennych
   integer :: nfact = 1
   integer :: n = 1

   ! oblicz kolejne wartości silni
   do while (n <= 10)
      nfact = nfact * n
      print*,  n, " ", nfact
      n = n + 1
   end do
end program silnia2
```

```
Overwriting silnia2.f90
```

In [29]:

```
!gfortran silnia2.f90 -o silnia2
!./silnia2
```

```
           1             1
           2             2
           3             6
           4            24
           5           120
           6           720
           7          5040
           8         40320
           9        362880
          10       3628800
```

Zagnieżdżone pętle

In [30]:

```
%%writefile nested.f90
program nested
implicit none

   integer:: i, j, k

   iloop: do i = 1, 3
      jloop: do j = 1, 3
         kloop: do k = 1, 3

            print*, "(i, j, k): ", i, j, k

         end do kloop
      end do jloop
   end do iloop

end program nested
```

```
Writing nested.f90
```

In [31]:

```
!gfortran nested.f90 -o nested
!./nested
```

```
 (i, j, k):            1           1           1
 (i, j, k):            1           1           2
 (i, j, k):            1           1           3
 (i, j, k):            1           2           1
 (i, j, k):            1           2           2
 (i, j, k):            1           2           3
 (i, j, k):            1           3           1
 (i, j, k):            1           3           2
 (i, j, k):            1           3           3
 (i, j, k):            2           1           1
 (i, j, k):            2           1           2
 (i, j, k):            2           1           3
 (i, j, k):            2           2           1
 (i, j, k):            2           2           2
 (i, j, k):            2           2           3
 (i, j, k):            2           3           1
 (i, j, k):            2           3           2
 (i, j, k):            2           3           3
 (i, j, k):            3           1           1
 (i, j, k):            3           1           2
 (i, j, k):            3           1           3
 (i, j, k):            3           2           1
 (i, j, k):            3           2           2
 (i, j, k):            3           2           3
 (i, j, k):            3           3           1
 (i, j, k):            3           3           2
 (i, j, k):            3           3           3
```

Funkcje i procedury

- **funkcja** to grupa poleceń, przyjmująca zestaw argumentów na wejściu i zwracająca jedną wartość na wyjściu
- **procedura** to grupa poleceń, która wprawdzie nie zwraca wartości, ale może modyfikować argumenty wywołania

Funkcje

In [117]:

```
%%writefile functions.f90
program func

   real :: a
   a = area_of_circle(2.0)

   print *, "Pole koła o promieniu 2 wynosi: ", a

end program func


! ta funkcja oblicza pole koła o promieniu r

function area_of_circle (r)
   implicit none

   real :: area_of_circle   !tu określany jest typ zwracanej wartości

   ! zmienne lokalne
   real :: r
   real :: pi

   pi = 4 * atan (1.0)
   area_of_circle = pi * r**2

end function area_of_circle
```

```
Writing functions.f90
```

In [118]:

```
!gfortran functions.f90 -o functions
!./functions
```

```
 Pole koła o promieniu 2 wynosi:    12.5663710
```

Procedury

In [119]:

```
%%writefile subroutines.f90
program subrou
implicit none

   real :: a, b
   a = 2.0
   b = 3.0

   print *, "Przed wywołaniem swap"
   print *, "a = ", a
   print *, "b = ", b

   call swap(a, b)

   print *, "Po wywołaniu swap"
   print *, "a = ", a
   print *, "b = ", b

end program subrou


subroutine swap(x, y)
implicit none

   real :: x, y, temp

   temp = x
   x = y
   y = temp

end subroutine swap
```

```
Writing subroutines.f90
```

In [120]:

```
!gfortran subroutines.f90 -o subroutines
!./subroutines
```

```
 Przed wywołaniem swap
 a =    2.00000000
 b =    3.00000000
 Po wywołaniu swap
 a =    3.00000000
 b =    2.00000000
```

Tablice

- elementy muszą być takiego samegu typu
- pierwszy element ma domyślnie indeks 1 !!!
- zakresy indeksów można przedefiniować
- jednowymiarowe (wektory), dwuwymiarowe (macierze), wielowymiarowe (max. 7 wymiarowe)

In [59]:

```
%%writefile arrays.f90
program arrays

   real, dimension(-2:6) :: strange ! tablica 1d o indeksach od -2 do 6 (włączni
e)

   real :: numbers(5) !jednowymiarowa tablica liczb rzeczywistych
   integer :: matrix(3,3), i , j !dwuwymiarowa tablica liczb całkowitych


   do i=1,5
      numbers(i) = i * 2.0
   end do

   print*, 'numbers:'
   do i = 1, 5
      print *, numbers(i)
   end do

   do i=1,3
      do j = 1, 3
         matrix(i, j) = i+j
      end do
   end do

   print*, 'matrix:'
   do i=1,3
      do j = 1, 3
         print *, matrix(i,j)
      end do
   end do

   !wersja skrócona
   numbers = (/1.5, 3.2,4.5,0.9,7.2 /)

   print*,'numbers dla leniwych:'
   print *, numbers

   !i tablica o 'dziwnych' indeksach
   do i=-2,6
      strange(i) = i * 2.0
   end do

   print*,'strange:'
   do i=-2,6
      print*,i,strange(i)
   end do
end program arrays
```

```
Overwriting arrays.f90
```

In [60]:

```
!gfortran arrays.f90 -o arrays
!./arrays
```

```
 numbers:
   2.00000000
   4.00000000
   6.00000000
   8.00000000
   10.0000000
 matrix:
           2
           3
           4
           3
           4
           5
           4
           5
           6
 numbers dla leniwych:
   1.50000000       3.20000005       4.50000000      0.899999976
    7.19999981
 strange:
          -2  -4.00000000
          -1  -2.00000000
           0   0.00000000
           1   2.00000000
           2   4.00000000
           3   6.00000000
           4   8.00000000
           5   10.0000000
           6   12.0000000
```

Wycinki tablic

In [65]:

```
%%writefile arrays2.f90
program arrays2

   real, dimension(10) :: a, b
   integer:: i, asize, bsize

   a(1:7) = 5.0 ! od a(1) do a(7) wstawiamy 5.0
   a(8:) = 0.0  ! pozostałe elementy a to 0.0
   b(2:10:2) = 3.9
   b(1:9:2) = 2.5

   asize = size(a)
   bsize = size(b)

   print*,'a:'
   do i = 1, asize
      print *, a(i)
   end do

   print*,'b:'
   do i = 1, bsize
      print *, b(i)
   end do

end program arrays2
```

```
Overwriting arrays2.f90
```

In [66]:

```
!gfortran arrays2.f90 -o arrays2
!./arrays2
```

```
 a:
   5.00000000
   5.00000000
   5.00000000
   5.00000000
   5.00000000
   5.00000000
   5.00000000
   0.00000000
   0.00000000
   0.00000000
 b:
   2.50000000
   3.90000010
   2.50000000
   3.90000010
   2.50000000
   3.90000010
   2.50000000
   3.90000010
   2.50000000
   3.90000010
```

Działania na tablicach

.. . .

In [67]:

```
%%writefile skalarny.f90
program skalarny

   real, dimension(5) :: a, b
   integer:: i, asize, bsize

   asize = size(a)
   bsize = size(b)

   do i = 1, asize
      a(i) = i
   end do

   do i = 1, bsize
      b(i) = i*2
   end do

   print *, 'a: ', a
   print *, 'b: ', b

   print*, 'Wynik mnożenia: ',dot_product(a, b)

end program skalarny
```

```
Writing skalarny.f90
```

In [68]:

```
!gfortran skalarny.f90 -o skalarny
!./skalarny
```

```
 a:    1.00000000       2.00000000       3.00000000       4.0000000
0       5.00000000
 b:    2.00000000       4.00000000       6.00000000       8.0000000
0       10.0000000
 Wynik mnożenia:    110.000000
```

Mnożenie macierzowe

In [85]:

```
%%writefile macierzowe.f90
program macierzowe

   integer, dimension(3,3) :: a, b
   integer :: i, j

   do i = 1, 3
      do j = 1, 3
         a(i, j) = i+j
      end do
   end do

   print *, 'a:'
   print '(3(I2,X))', a

   do i = 1, 3
      do j = 1, 3
         b(i, j) = i*j
      end do
   end do

   print *, 'b:'
   print '(3(I2,X))', b

   print *, 'Wynik mnożenia:'
   write(*,'(3(I2,X))'), matmul(a, b)
end program macierzowe
```

```
Overwriting macierzowe.f90
```

In [86]:

```
!gfortran macierzowe.f90 -o macierzowe
!./macierzowe
```

```
 a:
 2  3  4
 3  4  5
 4  5  6
 b:
 1  2  3
 2  4  6
 3  6  9
 Wynik mnożenia:
20 26 32
40 52 64
60 78 96
```

Redukcje

In [97]:

```
%%writefile redukcje.f90
program redukcje

   real, dimension(3,2) :: a
   a = reshape( (/5,9,6,10,8,12/), (/3,2/) )

   print '(3(F5.2,X))',a

   print *,'Suma elementów:', sum(a)

   print *,'Wszystkie elementy większe od 5?', all(a>5)
   print *,'Przynajmniej jeden element większy od 5?', any(a>5)
   print *,'Liczba elementów większych od 5', count(a>5)
   print *,'Czy wszystkie elementy większe równe 5 i mniejsze od 10', all(a>=5 .
and. a<10)

end program redukcje
```

```
Overwriting redukcje.f90
```

In [98]:

```
!gfortran redukcje.f90 -o redukcje
!./redukcje
```

```
 5.00  9.00  6.00
10.00  8.00 12.00
 Suma elementów:   50.0000000
 Wszystkie elementy większe od 5? F
 Przynajmniej jeden element większy od 5? T
 Liczba elementów większych od 5           5
 Czy wszystkie elementy większe równe 5 i mniejsze od 10 F
```

Tablice dynamiczne

In [99]:

```
%%writefile darrays.f90
program darrays
implicit none

   !tablica dwuwymiarowa, ale o nieznanych zakresach
   real, dimension (:,:), allocatable :: darray
   integer :: s1, s2
   integer :: i, j

   print*, "Wprowadź rozmiary tablicy 2D:"
   read*, s1, s2

   ! alokuj pamięć
   allocate ( darray(s1,s2) )

   do i = 1, s1
      do j = 1, s2
         darray(i,j) = i*j
         print*, "darray(",i,",",j,") = ", darray(i,j)
      end do
   end do

   !zwolnij pamięć
   deallocate (darray)
end program darrays
```

```
Writing darrays.f90
```

In [100]:

```
!gfortran darrays.f90 -o darrays
```

Łańcuchy znaków

In [50]:

```
%%writefile greetings.f90
program hello
implicit none

   character(len = 15) :: surname, firstname
   character(len = 6) :: title
   character(len = 40):: name
   character(len = 25)::greetings

   title = 'Mr. '
   firstname = 'Rowan '
   surname = 'Atkinson'

   name = title//firstname//surname
   greetings = 'A big hello from Mr. Bean'

   print *, 'Here is ', name
   print *, greetings

end program hello
```

```
Overwriting greetings.f90
```

In [51]:

```
!gfortran greetings.f90 -o greetings
!./greetings
```

```
 Here is Mr.   Rowan          Atkinson
 A big hello from Mr. Bean
```

In [52]:

```
%%writefile greetings2.f90
program hello2
implicit none

   character(len = 15) :: surname, firstname
   character(len = 6) :: title
   character(len = 40):: name
   character(len = 25)::greetings

   title = 'Mr. '
   firstname = 'Rowan '
   surname = 'Atkinson'

   name = trim(title)//trim(firstname)//trim(surname)
   greetings = 'A big hello from Mr. Bean'

   print *, 'Here is ', name
   print *, greetings

end program hello2
```

```
Writing greetings2.f90
```

In [53]:

```
!gfortran greetings2.f90 -o greetings2
!./greetings2
```

```
 Here is Mr.RowanAtkinson
 A big hello from Mr. Bean
```

Wybrane funkcje

Funkcja	Opis
`len(string)`	długość łańcucha znaków
`index(string,substring)`	położenie łańcucha `substring` w łańcuchu `string` (0 - brak)
`achar(int)`	przekształca liczbę całkowitą na znak
`iachar(c)`	przekształca znak na liczbę całkowitą
`trim(string)`	usuwa puste znaki na końcu łańcucha
`scan(string,chars)`	przeszukuje `string` do pierwszego trafienia dowolnego znaku z `chars`
`verify(string,chars)`	przeszukuje `string` do pierwszego trafienia dowolnego znaku spoza `chars`
`adjustl(string)`	justowanie łańcucha do lewej
`adjustr(string)`	justowanie łańcucha do prawej
`len_trim(string)`	długość łańcucha po usunięciu pustych znaków z jego końca
`repeat(string,ncopy)`	powtarza łańcuch `ncopy` razy

Struktury danych

In [111]:

```
%%writefile biblio.f90
program biblio

   !deklaracja struktury
   type Books
      character(len = 50) :: title
      character(len = 50) :: author
      character(len = 150) :: subject
      integer :: book_id
   end type Books

   !deklaracja zmiennych nowego typu
   type(Books) :: book1
   type(Books) :: book2

   !dostęp do elementów struktury

   book1%title = "C Programming"
   book1%author = "Nuha Ali"
   book1%subject = "C Programming Tutorial"
   book1%book_id = 6495407

   book2%title = "Telecom Billing"
   book2%author = "Zara Ali"
   book2%subject = "Telecom Billing Tutorial"
   book2%book_id = 6495700

   !wyświetl na ekranie

   print *,'Book1:'
   print *, book1%title
   print *, book1%author
   print *, book1%subject
   print *, book1%book_id

   print *,'Book2:'
   print *, book2%title
   print *, book2%author
   print *, book2%subject
   print *, book2%book_id

end program biblio
```

```
Overwriting biblio.f90
```

In [112]:

```
!gfortran biblio.f90 -o biblio
!./biblio
```

```
 Book1:
 C Programming
 Nuha Ali
 C Programming Tutorial


     6495407
 Book2:
 Telecom Billing
 Zara Ali
 Telecom Billing Tutorial


     6495700
```

Wskaźniki

- wskaźniki w Fortranie zawiera m.in. adres wybranego obiektu w pamięci
- może wskazywać na
  - obszar dynamicznie alokowanej pamięci
  - obiekt odpowiedniego typu

In [115]:

```
%%writefile pointers.f90
program pointers
implicit none

   integer, pointer :: p1
   integer, target :: t1
   integer, target :: t2

   !stowarzyszenie wskaźnika ze zmienną t1
   p1=>t1
   p1 = 1

   print *, p1
   print *, t1

   p1 = p1 + 4
   print *, p1
   print *, t1

   t1 = 8
   print *, p1
   print *, t1

   nullify(p1) !p1 przestaje wskazywać na zmienną p1
   print *, t1

   p1=>t2
   print *, associated(p1)
   print*, associated(p1, t1)
   print*, associated(p1, t2)

   !jaką wartość ma obecnie t2
   print *, p1
   print *, t2

   p1 = 10
   print *, p1
   print *, t2
end program pointers
```

```
Overwriting pointers.f90
```

In [116]:

```
!gfortran pointers.f90 -o pointers
!./pointers
```

```
           1
           1
           5
           5
           8
           8
           8
 T
 F
 T
   222603088
   222603088
          10
          10
```

Moduły

- z reguły zawierają funkcje i procedury do użytku w różnych programach
- pozwalają rozdzielić kod programu na wiele plików

In [121]:

```
%%writefile modules.f90
module constants
implicit none

   real, parameter :: pi = 3.1415926536
   real, parameter :: e = 2.7182818285

contains
   subroutine show_consts()
      print*, "Pi = ", pi
      print*,  "e = ", e
   end subroutine show_consts

end module constants


program module_example
use constants
implicit none

   real :: x, ePowerx, area, radius
   x = 2.0
   radius = 7.0
   ePowerx = e ** x
   area = pi * radius**2

   call show_consts()

   print*, "e do potęgi  2.0 = ", ePowerx
   print*, "Pole koła o promieniu 7.0 = ", area

end program module_example
```

```
Writing modules.f90
```

In [122]:

```
!gfortran modules.f90 -o modules
!./modules
```

```
 Pi =    3.14159274
 e =    2.71828175
 e do potęgi  2.0 =    7.38905573
 Pole koła o promieniu 7.0 =    153.938049
```