

Tablice

Tablice (ang. *array*) służą do przechowywania pewnej liczby wartości określonego typu i utożsamiania ich z jedną nazwą (w odróżnieniu od zmiennych prostych, w których jednej nazwie zmiennej odpowiada jedna wartość). Dostęp do poszczególnych elementów tablicy jest możliwy poprzez użycie indeksów. Każda tablica posiada typ (np. `REAL`, `INTEGER`); wszystkie elementy tablicy mają ten sam typ.

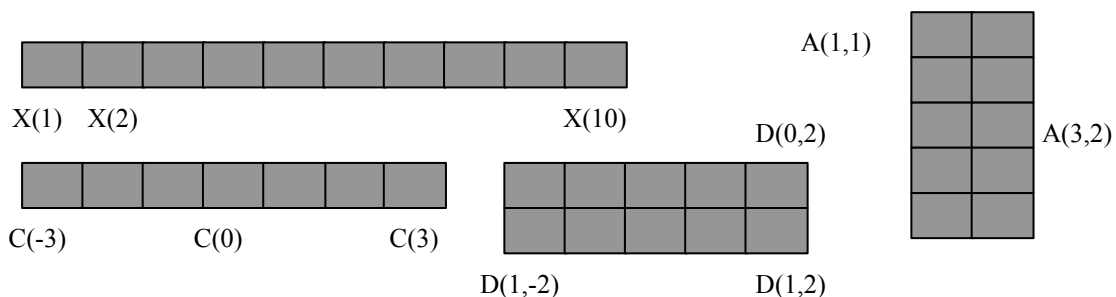
Deklarowanie tablic

Tablice muszą zostać zadeklarowane. Przykłady deklaracji:

```
REAL, DIMENSION (10) :: X
INTEGER, DIMENSION (1:5, 1:2) :: A
REAL :: C(-3:3), D(0:1, -2:2)
```

Zadeklarowano cztery tablice:

- Tablica `X` o elementach rzeczywistych jest tablicą jednowymiarową (wektorem), bowiem podano jeden zakres indeksów, o 10 elementach ponumerowanych od 1 do 10 (zapis `DIMENSION (10)` jest równoważny `DIMENSION (1:10)`);
- Tablica `A` jest tablicą dwuwymiarową (macierzą), o pięciu wierszach i dwóch kolumnach.
- Tablica `C` jest wektorem o siedmiu elementach rzeczywistych
- Tablica `D` jest macierzą o dwóch wierszach i pięciu kolumnach.



Terminologia

wymiar (ang. *rank*) – wymiar tablicy. Tablice `X` i `C` mają wymiar 1, zaś tablice `A` i `D` mają wymiar 2. Maksymalny wymiar tablicy to 7.

granice (ang. *bounds*) – najmniejsza i największa wartość przyjmowana przez indeksy; brak dolnej granicy w deklaracji oznacza wartość 1.

zakres (ang. *extent*) – liczba elementów dla danego wymiaru; w szczególności może być zerem. Zakres tablicy `X` wynosi 10, tablicy `C` – 7, tablicy `A` 5 i 2, zaś tablicy `D` – 2 i 5.

kształt (ang. *shape*) jest wektorem (a zatem również tablicą!) zawierającym zakresy tablicy. `X` ma kształt $(/10/)$, `C` – $(/7/)$, `A` – $(/5, 2/)$, `D` – $(/2, 5/)$ (sposób zapisu kształtu nie jest przypadkowy!)

rozmiar (ang. *size*) całkowita liczba elementów w tablicy (równa iloczynowi zakresów). W szczególności rozmiar tablic `A`, `D` i `X` jest taki sam i wynosi 10.

zgodność (ang. *conform*) – tablice o takim samym kształcie nazywa się zgodnymi lub konforemnymi. Najczęściej wymaga się zgodności tablic (lub ich wycinków) w operacjach, w których występują różne tablice (lub ich wycinki). Wartości skalarne są zgodne z tablicami o dowolnym kształcie.

Deklaracje – symboliczne zakresy

Dopuszczalne jest wykorzystanie w deklaracjach tablic zmiennych symbolicznych (`PARAMETER`), na przykład:

```
INTEGER, PARAMETER :: N=2
```

```
REAL, DIMENSION (10) :: X
INTEGER, DIMENSION (1:5, 1:N) :: A
REAL :: C(-3:N+1), D(0:1, -2:N)
```

Element tablicy

Element tablicy jest jednoznacznie zdefiniowany poprzez indeksy. Liczba indeksów jest równa wymiarowi tablicy (dla wektora – jeden indeks, dla tablicy dwuwymiarowej – dwa indeksy, itd.); poszczególne indeksy są oddzielane przecinkami. Indeks musi mieć typ całkowity, w ogólnym przypadku jest to wyrażenie arytmetyczne. Wartości indeksów muszą zawierać się przedziałach zdefiniowanych przez granice dolną i górną.

Wycinki tablic (ang. *section*)

Można odwoływać się ‘na raz’ do kilku elementów tablicy, nazywanych wycinkiem. Wycinki definiuje się za pomocą trójki indeksowej (ang. *subscript triplet*):

```
tablica ([zakres_dolny]:[zakres_górny][:krok], ...)
```

Użycie tych trzech wartości jest opcjonalne. Wartości przyjmowane domyślnie to granice wynikające z deklaracji, dla wartości `zakres_dolny` oraz `zakres_górny` oraz 1 dla kroku.

Przykład:

Jeśli:

```
INTEGER, DIMENSION(6) :: A
REAL, DIMENSION(2,4) :: C
```

Wtedy:

`A(:)` cała tablica

`A(::2)` elementy `A(1)`, `A(3)`, `A(5)`

`A(2:4:2)` elementy `A(2)`, `A(4)`

`A(3:)` elementy `A(3)`, `A(4)`, `A(5)`, `A(6)`

`C(:, 2::2)` wszystkie elementy z drugiej i czwartej kolumny

`C(:, 3)` wszystkie elementy z trzeciej kolumny

`C(1, ::2)` elementy `C(1,1)`, `C(1,3)` (co drugi element z pierwszego wiersza)

`A(4:2)` – wycinek o rozmiarze zero

`A(4:2:-1)` – elementy `A(4)`, `A(3)` i `A(2)`

Uwagi:

- Wycinek jest tablicą, w której dolna granica zawsze jest równa 1, bez względu na to, jakie indeksy obowiązywały w oryginalnej tablicy.
- Należy zwrócić uwagę na istotną różnicę pomiędzy `A(1)` oraz `A(1:1)`. `A(1)` jest elementem skalarnym, odwołaniem do elementu tablicy `A` o indeksie 1. Natomiast `A(1:1)` jest jednoelementowym wycinkiem.
- W definicji wycinków dopuszczalne jest wykorzystanie ujemnego kroku.

Indeksy wektorowe

Indeksy wektorowe (ang. *vector subscript*) są jednowymiarowymi tablicami całkowitymi zapisanymi w postaci:

```
(/lista/)
```

gdzie `lista` oznacza listę indeksów podanych w dowolnej kolejności. Umożliwiają odwoływanie się do grup elementów tablic, o ‘nieregularnym’ rozkładzie indeksów, który nie daje się opisać za pomocą wycinków.

Przykład:

```
REAL, DIMENSION(10) :: A
```

```

REAL, DIMENSION(3) :: C
INTEGER :: LISTA(5)
INTEGER, DIMENSION(3) :: V=(/1,5,3/), W=(/2,4,2/)
! wartości indeksów wektorowych określone w deklaracjach...
lista = (/4,8,9,1,3/)
! ... oraz w instrukcji przypisania.
A = 2.
A (lista) = 0.
! przypisanie wartości 0. elementom A(4), A(8), A(9), A(1) i A(3)
C=A(W)
! równoważne: C(1)=A(2), C(2)=A(4), C(3)=A(2)
A(V)=C
! równoważne: A(1)=C(1), A(5)=C(2), A(3)=C(3)

```

- Indeksy wektorowe mogą być wykorzystywane zarówno po prawej jak i po lewej stronie instrukcji przypisania. Należy jednak przestrzegać zasady, aby w przypadku gdy występują po lewej stronie instrukcji przypisania występujące w nich indeksy były jednoznaczne (tego warunku nie spełnia wektor w).
- Zakresy indeksów występujące w indeksach wektorowych muszą zawierać się w przedziałach zdefiniowanych poprzez granice dolne i górne.
- Indeksy wektorowe są bardzo niewydatne i nie należy ich używać bez potrzeby.

Kolejność elementów w tablicy

Standard Fortranu 90 (w przeciwieństwie do standardu Fortranu 77) nie definiuje w jaki sposób tablice są przechowywane w pamięci komputera.

Jednak są sytuacje, w których niezbędne jest zdefiniowanie uporządkowania (np. podczas wczytywania danych lub wypisywania wyników). Obowiązuje tu kolejność znana z poprzednich wersji Fortranu: elementy tablicy są uporządkowane w kolejności określonej najszybszą zmianą pierwszego indeksu, co w przypadku tablic dwuwymiarowych (macierzy) oznacza, że elementy są uporządkowane ‘kolumnami’.

Tak zdefiniowane uporządkowanie oprócz wspomnianych już operacji wejścia i wyjścia jest wykorzystywane przez konstruktory, niektóre funkcje standardowe (np. TRANSFER, RESHAPE, PACK, UNPACK i MERGE) oraz wszędzie tam gdzie niezbędne jest liniowe uporządkowanie elementów tablicy.

Tablice: Instrukcje wejścia/wyjścia

Podobnie jak w przypadku zmiennych prostych, instrukcja czytania (READ) umożliwia wprowadzenie ‘z zewnątrz’ wartości i nadanie ich elementom tablicy, natomiast instrukcja pisania (WRITE lub PRINT) pozwala wartości elementów tablicy wypisać.

Przykład:

```
REAL, DIMENSION(10) :: A
```

Instrukcja

```
READ*, A
```

oznacza wczytanie z klawiatury 10 wartości, które będą podstawiane kolejno pod A(1), A(2), itd. Kolejne wartości należy oddzielać przecinkiem, spacją lub znakiem nowej linii. Liczba czytanych wartości jest zgodna z rozmiarem tablicy.

W przypadku zaprogramowania czytania w postaci:

```
DO I=1, 10
  READ*, A(i)
ENDDO
```

sytuacja jest podobna, natomiast kolejne wartości muszą być wprowadzane od nowego wiersza (każda instrukcja READ rozpoczyna czytanie od nowej linii).

W instrukcji czytania można wykorzystywać wycinki; na przykład:

```
READ*, A(:,2)
```

oznacza, wczytanie pięciu wartości, które będą podstawiane kolejno na $A(1)$, $A(3)$, ..., $A(9)$.

DO implikowane

Konstrukcja 'DO-implikowane' jest wykorzystywana w wielu sytuacjach związanych z tablicami. Jest analogiczna do postaci pętli DO indeksowanej:

```
READ*, (A(I), I=1, 10)
```

oznacza wczytanie dziesięciu wartości, które będą podstawiane kolejno pod $A(1)$, $A(2)$, itd.

Instrukcja:

```
READ*, (A(I), I=10, 5, -1)
```

oznacza wczytanie sześciu wartości i podstawianie ich kolejno pod $A(10)$, $A(9)$, ..., $A(5)$.

Wszystkie podane tu informacje obowiązują również dla instrukcji pisania PRINT oraz WRITE.

Na przykład:

```
PRINT*, A
```

spowoduje wydrukowanie *wszystkich* elementów tablicy A, czyli 10 wartości, odpowiadających $A(1)$, $A(2)$, itd.

W przypadku ciągu instrukcji w postaci:

```
DO I=1, 10  
  PRINT*, A(i)  
ENDDO
```

wartości kolejnych elementów zostaną wypisane od nowego wiersza (bowiem każda instrukcja PRINT i WRITE rozpoczyna pisanie od nowej linii).

W instrukcji pisania można wykorzystywać wycinki. Na przykład:

```
PRINT*, A(:,2)
```

oznacza wypisanie pięciu wartości odpowiadających elementom $A(1)$, $A(3)$, ..., $A(9)$.

Konstrukcja DO-implikowane również jest bardzo przydatna podczas wypisywania:

```
PRINT*, (A(I), I=1, 10)
```

oznacza wypisanie wartości, odpowiadających $A(1)$, $A(2)$, itd.

Instrukcja:

```
PRINT*, (A(I), I=10, 5, -1)
```

oznacza wypisanie sześciu wartości odpowiadających $A(10)$, $A(9)$, ..., $A(5)$.

Czytanie – tablice dwuwymiarowe

Kolejność elementów w przypadku tablic jednowymiarowych nie budzi wątpliwości. W przypadku tablic dwuwymiarowych (lub o większym wymiarze) sprawa jest bardziej skomplikowana.

Na przykład, rozważamy dwuwymiarową tablicę A o 2 wierszach i 3 kolumnach:

```
INTEGER, DIMENSION (2,3) :: A
```

Założmy, że chcemy aby jej elementy miały wartości:

```
1, 3, 5  
2, 4, 6
```

Instrukcja czytania

```
READ, * A
```

wymaga podania sześciu wartości (2x3), które będą przypisywane elementom w kolejności $A(1,1)$, $A(2,1)$, $A(1,2)$, $A(2,2)$, $A(1,3)$, $A(2,3)$. Wykorzystanie instrukcji READ jest jedną z sytuacji, kiedy niezbędna jest znajomość kolejności elementów w tablicy obowiązująca w Fortranie (patrz punkt: *Kolejno elementów w tablicach*). Dla tablic dwuwymiarowych obowiązuje kolejność *kolumnami*.

Metodą umożliwiającą wczytywanie elementów tablicy w innej kolejności jest wykorzystanie pętli DO oraz wycinka:

```
DO I=1,2
  READ*, A(i,:) ! wszystkie elementy i-tego wiersza
ENDDO
```

lub też zastosowanie DO-implikowanego do kolejnych wierszy:

```
DO I=1,2
  READ*, (A(i,j) , j=1,3) ! wszystkie elementy i-tego wiersza
ENDDO
```

Obydwie konstrukcje pozwolą uzyskać ten sam efekt: proces pobierania danych przez program przebiegać będzie w kolejności zdefiniowanej wierszami, dla każdego wiersza (od 1 do 2) czytane będą wszystkie (po 3) należące do niego elementy.

Jeszcze innym sposobem umożliwiającym wczytanie elementów w kolejności zgodnej z wierszami jest wykorzystanie zagnieżdżonych konstrukcji DO-implikowanego:

```
READ*, ((A(i,j) , j=1,3) , i=1,2)
```

Analogicznie do zagnieżdżonych instrukcji pętli, indeksem 'szybciej' się zmieniającym jest indeks wewnętrzny, a zatem numer kolumny, przy ustalonym indeksie zewnętrznym odpowiadającym numerowi wiersza.

Zmiana kolejności w zagnieżdżeniu:

```
READ*, ((A(i,j) , i=1,2) , j=1,3)
```

spowoduje, że kolejne elementy będą podstawiane kolumnami.

Pisanie – tablice dwuwymiarowe

Podobnie rzecz się ma podczas wypisywania wartości tablic. Instrukcja:

```
PRINT*, A
```

spowoduje wypisanie ciągu liczb

```
1, 2, 3, 4, 5, 6
```

odpowiadających kolejnym (czyli uporządkowanym wzdłuż kolumn) wartościom elementów tablicy A. Taka postać instrukcji nie umożliwia wypisania elementów w postaci 'naturalnej', gdzie kolejne wiersze są wypisywane od nowej linii. Taki efekt można osiągnąć korzystając z jawnej pętli DO:

```
DO I=1,2
  PRINT*, (A(i,j) , j=1,3) ! wszystkie elementy i-tego wiersza
ENDDO
```

Rozbudowaną wersję tej instrukcji stanowi kolejny przykład:

```
DO I=1,2
  PRINT*, 'wiersz', i, ':' (A(i,j) , j=1,3)
ENDDO
```

w którym dodatkowo na początku każdego wiersza pojawi się słowo 'wiersz' wraz z numerem wiersza.

Instrukcja wykorzystująca zagnieżdżoną konstrukcję DO-implikowanego

```
PRINT*, ((A(i,j) , j=1,3) , i=1,2)
```

wprawdzie wypisze wartości kolejnych elementów w kolejności ‘wierszami’, ale nie rozdzieli wierszy macierzy linie wydruku.

Tablice – przykłady instrukcji przypisania

W instrukcjach przypisania mogą występować:

– odwołania do elementów tablic:

$A(1) = 0.$

$B(2, 3) = A(3) + C(2, 4)$

Uwaga:

Indeksy muszą być wartościami o typie `INTEGER` (stałe, zmienne, wyrażenia).

– odwołania do całych tablic:

$A=0.0$

$A(:, :)=0.0$ (zapis równoważny do pierwszego; tym razem widać, że A to tablica dwuwymiarowa).

$B=C+D$

$F=G*H$

$B=2.+ A$

W wyniku wykonania tych instrukcji odpowiadające sobie elementy tablic C i D zostaną dodane, a wynik umieszczony w odpowiednich miejscach tablicy B . Analogicznie, elementy tablic G i H są mnożone. Wartości elementów tablicy B są równe odpowiadającym im elementom tablicy A plus *warto skalarna 2*.

Uwagi:

- W przypadku niezgodności *typów* tablic następuje konwersja zgodnie z obowiązującymi zasadami.
- Tablice uczestniczące w operacji (B, C i D, F, G i H, B i A) muszą być zgodne, tzn. muszą mieć taki sam kształt i rozmiar.
- Wartość skalarna jest zgodna z tablicami o dowolnym kształcie.
- Z konceptualnego punktu widzenia, operacje na tablicach są wykonywane ‘równolegle’, dla każdego elementu niezależnie od pozostałych.
- **Nie jest to** równoważne algebraicznemu mnożeniu macierzy przez macierz (funkcja `MATMUL`).

– odwołania do wycinków

Analogicznie do operacji zdefiniowanych w odniesieniu do całych tablic, możliwe jest wykonywanie operacji na wycinkach. Nie ma w tym nic dziwnego, bowiem wycinki też są tablicami. Podobnie jak w przypadku tablic, uczestniczące w operacji wycinki muszą być zgodne (natomiast całe tablice, do których te wycinki należą, nie muszą być zgodne.)

Należy zwrócić uwagę na sytuację, gdy zarówno po prawej stronie, jak i po lewej stronie instrukcji przypisania występują wycinki należące do tej samej tablicy:

$A(2:10) = A(1:9)$

jest poprawną instrukcją. W wyniku jej wykonania nastąpi ‘przesunięcie’ (ang. *shift*) o jedną pozycję ‘w lewo’ dziewięciu elementów A .

Odpowiada to ciągowi instrukcji:

`DO i = 10, 2, -1`

`A(i) = A(i-1)`

`ENDDO`

(Zwracamy uwagę na kierunek wykonania pętli).

Rozważmy inną sytuację:

$A(2:10) = A(2:10) + A(1:9)$

oznacza, że należące do wynikowego wycinka elementy tablicy A będą miały wartości równe:

```
A(i) = A(i) + A(i-1)
```

Wykonanie na pozór analogicznej instrukcji:

```
DO i = 2, 10
  A(i) = A(i) + A(i-1)
ENDDO
```

spowoduje zupełnie inny skutek, spowodowany tym, że w obliczeniach wykonywanych dla kolejnych elementów będą wykorzystywane wartości elementów A, które zostały zmodyfikowane w poprzednich obiegach pętli.

Zmiana 'kierunku' realizacji pętli pozwala uzyskać wynik równoważny do operacji na wycinkach:

```
DO i = 10, 2, -1
  A(i) = A(i) + A(i-1)
ENDDO
```

Wykorzystanie matematycznych funkcji standardowych

Procedury, które do tej pory były stosowane w odniesieniu do argumentów skalarnych, mogą być wykorzystywane również wtedy, gdy ich parametrami są tablice.

```
REAL, DIMENSION(10) :: X, Y, C
REAL, DIMENSION(10,10) :: B, C, D
....
B=sin(C)+cos(D)
C=SQRT(X**2+Y**2)
```

Funkcje te są realizowane niezależnie dla każdego elementu tablicy; ponownie ma miejsce konceptualna równoległość.

Tablice zero-wymiarowe

Fortran 90 dopuszcza, aby wymiar tablicy był równy zero. Zdarza się tak wtedy, gdy zakres górny jest mniejszy od zakresu dolnego. Tablica zero-wymiarowa nie ma wartości (bo nie zawiera żadnego elementu), ale jest obiektem poprawnym i zawsze definiowalnym. Użyteczność takich tablic związana jest z bardziej zaawansowanymi zastosowaniami.

Inicjalizowanie tablic

Konstruktory (ang. *constructor*)

Nadawanie tablicom jednowymiarowym wartości początkowych może odbywać się za pomocą konstruktorów. Konstruktor jest listą ograniczoną ukośnikami i umieszczoną w nawiasach:

```
(/lista/)
```

gdzie *lista* może być:

- listą wartości odpowiedniego typu

```
INTEGER :: A(6)=(/1,2,3,6,7,8/)
```
- listą wyrażeń

```
REAL :: B(2)=(/sin(X), cos(X)/)
```
- wyrażeniem tablicowym

```
INTEGER, DIMENSION(5) :: C=(/0,A(1:3),4/)
```
- pętlą implikowaną DO

```
INTEGER :: D(50)=(/ (I, I=1,50) /)
```

Konstruktory mogą być wykorzystane w deklaracjach oraz w instrukcjach podstawienia.

Przykłady deklaracji:

```
INTEGER, DIMENSION(3), PARAMETER :: Unit_vec(/1,1,1/)
CHARACTER (LEN=*), DIMENSION(3), PARAMETER :: &
  kolory = (/ 'NIEBIESKI', 'CZERWONY ', 'ZIELONY ' /)
REAL, DIMENSION(2,2), PARAMETER :: &
```

```
Unit_matrix = RESHAPE ((/1.,0.,0.,1./), (/2,2/))
```

Ponieważ konstruktory można stosować tylko w odniesieniu do tablic jednowymiarowych, w przypadku tablic o większym wymiarze konstruktory używane są w połączeniu z funkcjami zmieniającymi kształt tablicy, np `RESHAPE()`.

Instrukcja DATA

Innym sposobem inicjalizowania tablic jest instrukcja DATA:

DATA zmienna /lista/

Omawiamy tę instrukcję w kontekście tablic, ale może być ona również stosowana do nadawania wartości zmiennym skalarnym.

Przykład:

```
INTEGER :: A(5), B(2,2), C(10), E(4)
DATA A /1,2,3,4,5/
! lista wartosci
DATA E /4*0/
! lista wartosci. 4*0 oznacza czterokrotne powtorzenie wartosci 0
DATA B(1,:) /0,0/ ! pierwszy wiersz tablicy B
DATA B(2,:) /2*1/ ! drugi wiersz tablicy B
DATA (C(i),i=1,10,2)/5*1/ DATA (C(i),i=2,10,2) /5*2/
```

W ostatniej instrukcji pojawiła się po raz kolejny konstrukcja DO-implikowane.

WHERE

Instrukcja przypisania `WHERE` umożliwia wykonanie tablicowej instrukcji podstawienia tylko dla wybranych elementów. Wybór elementów następuje na podstawie spełnienia warunku nazywanego maską (ang. *mask*).

`WHERE` (warunek) wyrażenie

Na przykład:

```
WHERE (A<0) A= -A
```

W wyniku wykonania tej instrukcji ujemnym elementom tablicy A zostanie zmieniony znak.

Można też stosować instrukcję `WHERE` w postaci blokowej:

```
WHERE (warunek)
  blok 1
ELSEWHERE
  blok2
ENDWHERE
```

Na przykład:

```
WHERE (B < 3)
  B = B+2
ELSEWHERE
  B = 0
ENDWHERE
```

Elementy tablicy B o wartościach mniejszych od 3 zostaną zwiększone o 2, zaś pozostałym elementom B zostanie nadana wartość 0.

Uwaga:

`WHERE` jest tablicową instrukcją przypisania i nie należy mylić jej z instrukcją warunkową. W szczególności nie może zawierać innych instrukcji, np. pisania.

Tablice dynamiczne

Fortran 90 pozwala na dynamiczne tworzenie tablic podczas wykonywania programu. Są to tablice ‘tymczasowe’ w tym sensie, że są tworzone, wykorzystywane oraz usuwane podczas wykonywania programu.

Tablice dynamiczne należy w odpowiedni sposób zadeklarować, a następnie zaalokować (ang. *allocate*). Po zakończeniu działań na takiej tablicy powinno zwolnić się zajmowaną przez nią pamięć (zdealokować, ang. *deallocate*).

Deklaracja tablic dynamicznych

W deklaracjach tablic dynamicznych pomija się rozmiar tablic; wymiar tablicy definiuje się za pomocą dwukropków:

```
INTEGER, DIMENSION(:), ALLOCATABLE :: X
REAL, DIMENSION(:, :), ALLOCATABLE :: A
```

Uaktywnianie tablic dynamicznych

Przydzielenie miejsca w pamięci i uaktywnienie tablicy dynamicznej następuje poprzez wykonanie instrukcji:

```
ALLOCATE (X(10))
albo
ALLOCATE (X(10), STAT=ierr)
if (ierr /= 0) then
    PRINT*, 'blad alokacji tablicy X'
endif
```

Sprawdzenie statusu operacji (*STAT*) pozwala zareagować na ewentualny błąd wykonania alokacji pamięci. W przykładzie *ierr* jest zmienną typu *INTEGER*. Wartość *ierr* jest równa 0 gdy alokacja przebiegła pomyślnie; w przeciwnym przypadku jest różna od zera.

Zwalnianie pamięci

Zwolnienie miejsca w pamięci następuje w wyniku wykonania instrukcji:

```
DEALLOCATE (X)
albo
if (ALLOCATE(X)) DEALLOCATE (X, STAT=ierr)
```

Uwaga:

Zarządzanie tablicami dynamicznymi jest droższe od zarządzania tablicami statycznymi. Dlatego też wszędzie tam, gdzie rozmiar tablic jest z góry znany oraz gdy tablice są wykorzystywane znaczącą część czasu realizacji programu, zaleca się stosować tablice statyczne.