

Instrukcje warunkowe

Instrukcje warunkowe, zgodnie z nazwą, umożliwiają wykonanie określonego ciągu instrukcji tylko wtedy, gdy spełniony jest zadany warunek. Fortran dysponuje kilkoma wariantami instrukcji warunkowych. Niezależnie od wariantu stosowanej instrukcji warunkowej, poprawne sformułowanie warunku ma podstawowe znaczenie dla uzyskania pożądanych efektów. Warunki wyraża się za pomocą wyrażeń logicznych przyjmujących wartości `.true.` (prawda) gdy warunek jest spełniony oraz `.false.` (fałsz) w przeciwnym przypadku.

Instrukcja IF – warunkowe wykonanie jednej instrukcji

```
IF (warunek) instrukcja
```

Np.

```
IF (i>15) write (*,*) 'wartosc jest wieksza od 15'
```

Działanie

Instrukcja `write (*,*) 'wartosc jest wieksza od 15'` zostanie zrealizowana tylko wtedy, gdy spełniony jest warunek `i>15`. W przeciwnym przypadku instrukcja ta nie zostanie wykonana.

Taka konstrukcja `IF` umożliwia warunkowe zrealizowanie dokładnie jednej instrukcji.

Instrukcja blokowa IF-THEN

```
IF (warunek) THEN
    instrukcja
    instrukcja
    ....
ENDIF
```

Np.

```
IF (x>0 .and. y>0) THEN
    d=sqrt(x*x+y*y)
    write (*,*) 'punkt znajduje sie w pierwszej&
                cwiartce układu współrzędnych'
ENDIF
```

Działanie:

Instrukcje znajdujące się między instrukcją `IF` a instrukcją `ENDIF` zostaną wykonane tylko wtedy gdy wartością wyrażenia logicznego warunek jest `.true.`

Instrukcja blokowa IF-THEN-ELSE

```
IF (warunek) THEN
    instrukcja
    instrukcja
    ....
ELSE
    instrukcja
    instrukcja
    ....
ENDIF
```

Działanie:

Instrukcje znajdujące się między instrukcją `IF` a instrukcją `ELSE` zostaną wykonane tylko wtedy gdy spełniony jest warunek. W przeciwnym przypadku zostaną wykonane instrukcje znajdujące się pomiędzy instrukcją `ELSE` i `ENDIF`.

Instrukcja blokowa IF- ELSE-ELSE IF

Instrukcje IF-THEN oraz IF-THEN-ELSE stanowią przypadki szczególne instrukcji:

```
IF (warunek) THEN
    instrukcja
    instrukcja
    ....
ELSE IF (warunek1) THEN
    instrukcja
    instrukcja
    ....
ELSE IF (warunek2) THEN
    instrukcja
    instrukcja
    ....
ELSE
    instrukcja
    instrukcja
    ....
ENDIF
```

Warunki (warunek, warunek1, warunek2, itd.) są rozpatrywane po kolei. Jeśli warunek jest spełniony, to po zrealizowaniu związanych z nim instrukcji następuje zakończenie wykonywania całej instrukcji warunkowej. W przeciwnym przypadku sprawdzane jest spełnienie kolejnego warunku.

Podczas realizacji instrukcji warunkowej zostają wykonane tylko instrukcje związane z pierwszym napotkanym spełnionym warunkiem.

Uwagi:

- Instrukcje warunkowe mogą się zagnieżdżać.
- Nie można ‘wskoczyć’ do wnętrza instrukcji warunkowej.
- Instrukcjom warunkowym można przypisywać nazwy, przy czym ma to znaczenie wyłącznie ‘kosmetyczne’ i zwiększające czytelność programu:

```
ala: IF (warunek) THEN
    instrukcja
    instrukcja
    ....
ELSE IF (warunek1) THEN
    instrukcja
    instrukcja
    ....
ELSE IF (warunek2) THEN
    instrukcja
    instrukcja
    ....
ELSE
    instrukcja
    instrukcja
    ....
ENDIF ala
```

Uwaga:

Nazwa musi być poprawną nazwą w rozumieniu języka Fortran, tzn. musi rozpoczynać się od litery, może zawierać cyfry i znak podkreślenia, itd.

Jeśli nazwa znajduje się w linii ENDIF, to musi wystąpić również w pierwszej linii przed słowem kluczowym IF.

Dodatkowo można umieszczać nazwę przy poszczególnych rozgałęzieniach, ale w takim przypadku ta sama nazwa musi pojawić się w pierwszej oraz ostatniej linii instrukcji warunkowej:

```

ala: IF (warunek) THEN
        instrukcja
        instrukcja
        ....
ELSE IF (warunek1) THEN ala
        instrukcja
        instrukcja
        ....
ELSE IF (warunek2) THEN ala
        instrukcja
        instrukcja
        ....
ELSE ala
        instrukcja
        instrukcja
        ....
ENDIF ala

```

Instrukcja SELECT CASE

Instrukcja `SELECT CASE` spełnia podobną rolę jak instrukcja warunkowa. Jest przydatna wtedy gdy algorytm wymaga wybrania jednej ze ‘ścieżek postępowania’ na podstawie wartości określonego wyrażenia.

Np.

```

SELECT CASE (i)           ! wyrażenie
  CASE (1, 5, 10)        ! zakres1 wartości wyrażenia
    instrukcja1
    instrukcja2
  CASE (20:)             ! zakres2 wartości wyrażenia
    instrukcja3
  CASE DEFAULT          ! pozostałe zakresy
    instrukcja4
END SELECT

```

Działanie:

Jeśli $i=1$ lub $i=5$ lub $i=10$ wykonywane są instrukcja1 oraz instrukcja2. Jeśli $i>20$ wykonywana jest instrukcja3. Instrukcja4 zostanie wykonana w przypadku niespełnienia żadnego z poprzedzających warunków.

Uwagi:

- Wyrażenie, w zależności od którego jest wykonywana instrukcja `SELECT CASE` musi mieć typ całkowity (`integer`), znakowy (`character`) lub logiczny (`logical`).
- Liczba gałęzi w instrukcji jest nieograniczona, natomiast może w niej wystąpić tylko jedna gałąź `CASE DEFAULT`.
- Po wykonaniu instrukcji należących do określonej gałęzi (dla której spełniony był warunek) następuje zakończenie wykonywania całej konstrukcji `SELECT CASE`.
- Jeśli nie został spełniony warunek dla żadnej gałęzi, następuje wykonanie instrukcji z gałęzi `CASE DEFAULT`, o ile taka gałąź istnieje (odpowiednik frazy `ELSE` w instrukcji warunkowej).
- Nie można ‘wskakiwać’ do wnętrza instrukcji `SELECT CASE`.
- Instrukcje `SELECT CASE` mogą być zagnieżdżone.
- Instrukcje `SELECT CASE` można nazywać, przy czym podobnie jak w przypadku instrukcji `IF` ma to znaczenie tylko wpływające na czytelność programu. Jeśli nazwę umieszczono w

linii rozpoczynającej `SELECT CASE (...)`, to musi się ona znaleźć również w wierszu `END SELECT`. Jeśli nazwana została którakolwiek z gałęzi, to ta sama nazwa musi pojawić się w wierszach rozpoczynających i zamykających.

```
ala: SELECT CASE (i)
      CASE (1, 5, 10)
        instrukcja 1
        instrukcja 2
      CASE (20:)
        instrukcja 3
      CASE DEFAULT
        instrukcja 4
END SELECT ala
```

albo:

```
ala: SELECT CASE (i)
      CASE (1, 5, 10) ala
        instrukcja 1
        instrukcja 2
      CASE (20:) ala
        instrukcja 3
      CASE DEFAULT ala
        instrukcja 4
END SELECT ala
```

Sposoby definiowania zakresów wartości wyrażenia determinujących realizację określonych gałęzi:

- jedna wartość, np. (1);
- kilka wartości oddzielonych przecinkiem, np. (1, 5, 10);
- zakres: dwie wartości oddzielone dwukropkiem, np. (5:10), definiuje przedział domknięty, do którego należy wartość wyrażenia;
- Brak podania pierwszej wartości oznacza „mniejsze/równe drugiej wartości”, np. (:0);
- brak podania drugiej wartości oznacza „większe/równe pierwszej wartości”, np. (20:).
- Powyższe elementy można łączyć, oddzielając je przecinkiem.

Na przykład:

```
SELECT CASE (i)
  CASE (:0)
    print *, 'i <= 0'
  CASE (2:5, 11:13)
    print*, 'i z przedzialu [2,5] lub [11,13]'
```

```
  CASE (15, 17, 19)
    print*, 'i=15 lub i=17 lub i=19'
```

```
  CASE (20:)
    print*, 'i>20'
```

```
  CASE DEFAULT
    print*, 'i=1, 6 ,7,8,9,10,16 lub18'
```

```
END SELECT
```

Konstrukcja `SELECT CASE` jest bardziej wydajna od instrukcji `ELSE-IF`. Wartość wyrażenia sterującego jej wykonaniem jest wyznaczana tylko raz, po czym następuje przejście do wykonywania instrukcji należących do odpowiedniej gałęzi.

Pętle – instrukcje cyklicznie powtarzane

W Fortranie 90 istnieje kilka różnych sposobów realizowania pętli (cyklu), ang. *loop*. Przez pętlę (cykl) rozumiemy instrukcję powodującą wielokrotne wykonanie ciągu instrukcji.

Instrukcja DO

Najprostsza pętla zapisuje się w Fortranie jako:

```
DO
  instrukcja1
  instrukcja2
  .....
END DO
```

```
ala: DO
  instrukcja1
  instrukcja2
  .....
END DO ala
```

Zauważmy, że może się zdarzyć, że taka pętla będzie się wykonywać w nieskończoność. Aby do tego nie doszło, wewnątrz instrukcji cyklu powinna znaleźć się instrukcja warunkowa, która w określonej sytuacji (spełnienie warunku) spowoduje zakończenie wykonywania pętli. Taka instrukcja warunkowa najczęściej uaktywnia instrukcję skoku (wyprowadzającą sterowanie poza obszar działania pętli) lub instrukcję `EXIT`, która przenosi realizację programu do pierwszej instrukcji znajdującej się po instrukcji zamykającej pętlę `ENDDO`.

Na przykład:

```
i = 0
DO
  i = i + 5
  print*, 'i=', i
  if ( i>10) EXIT
END DO
print *, 'petla sie skonczyla. i ma wartosc', i
```

Wykonanie powyższego ciągu instrukcji spowoduje wypisanie na monitorze następujących komunikatów:

```
i=5
i=10
i=15
petla sie skonczyla. i ma wartosc 15
```

Uwagi:

- Umieszczenie instrukcji `EXIT` poza blokiem pętli jest błędem.
- Nie wolno 'wskakiwać' do wnętrza pętli.
- Można zagnieżdżać pętle.

Drugą obok `EXIT` instrukcją, która może wystąpić wyłącznie w obrębie instrukcji cyklu, jest instrukcja `CYCLE`. Jej wystąpienie (które powinno być związane z instrukcją warunkową) powoduje przerwanie realizacji instrukcji wchodzących w skład zakresu instrukcji cyklu i przystąpienie do realizowania pętli począwszy od jej *pierwszej* instrukcji.

Na przykład:

```
i=0
DO
  i = i + 3
```

```

    if (i<8 .or. i>15) CYCLE
    if (i>20) EXIT
    print *, 'i=', i
ENDDO
print *, 'petla sie skonczyła. i ma wartosc', i

```

Wykonanie powyższego ciągu instrukcji spowoduje wypisanie na monitorze następujących komunikatów:

```

i=9
i=12
i=15
petla sie skonczyła. i ma wartosc 21

```

Wystąpienie instrukcji CYCLE sprawiło, że dla niektórych wartości zmiennej *i* nie doszło do realizacji instrukcji `print*, 'i=', i`, bowiem wcześniej nastąpiło przerwanie realizacji cyklu i rozpoczęcie pętli 'od nowa'.

Instrukcja DO-WHILE

Kolejna instrukcja cyklu, nazywana tutaj DO-WHILE, wymaga podania warunku determinującego jej wykonanie.

```

DO WHILE (warunek)
    instrukcja1
    instrukcja2
    .....
END DO

```

```

ala: DO (warunek)
    instrukcja1
    instrukcja2
    .....
END DO ala

```

Instrukcje wchodzące w zakres instrukcji DO-WHILE są cyklicznie tak długo, jak długo wartością zmiennej lub wyrażenia warunek jest `.true`.

Na przykład:

```

i = 0
DO WHILE (i<6)
    print *, 'i=', i
    i = i + 3
END DO
print *, 'koniec petli, i=', i

```

W wyniku wykonania tego ciągu instrukcji na monitorze pojawią się kolejno komunikaty:

```

i=0
i=3
koniec petli, i=6

```

Jak widać, pętla ta wykonała się dwa razy. Za trzecim razem warunek sterujący nie był spełniony (*i*=6 nie jest mniejsze od 6) i dlatego nastąpiło zakończenie jej działania.

Zauważmy też, że pętla:

```

DO WHILE (.true.)
    ....
    ....
END DO

```

jest pętlą nieskończoną, natomiast pętla

```

DO WHILE (.false.)
    ....
    ....
END DO

```

nie wykona się ani razu.

Indeksowana instrukcja DO

Zdefiniowane do tej pory instrukcje pozwalają tak skonstruować instrukcję cyklu, aby wykonała się określoną liczbę razy. Załóżmy, że instrukcja cyklu ma zostać wykonana 10 razy. Można to osiągnąć na przykład w następujący sposób:

```
! sytuacja 1
i=0
DO WHILE (i<10)
    ....
    i = i + 1
    ....
END DO
```

albo:

```
! sytuacja 2
i=5
DO WHILE (i<=50)
    ....
    i = i + 5
    ....
END DO
```

albo:

```
! sytuacja 3
i=10
DO
    ....
    i = i - 1
    IF (i < 1) EXIT
ENDDO
```

W każdym z przedstawionych przykładów kontrola liczby powtórzeń jest osiągnięta za pomocą systematycznie modyfikowanej wartości pewnej zmiennej (tu o nazwie *i*) oraz warunku tak powiązanego z wartościami przyjmowanymi przez *i* w kolejnych cyklach aby zapewnić liczbę powtórzeń równą 10. Sterowanie tego rodzaju, realizowane za pomocą pewnej zmiennej (nazywanej *zmienną sterującą*) można uzyskać stosując instrukcję cyklu nazywaną *DO-
indeksowanym*. Ma ona następującą postać:

```
DO zmienna sterująca = wartość początkowa, wartość końcowa [, krok]
    instrukcja1
    instrukcja2
    ...
ENDDO
```

Znaczenie oznaczeń jest następujące:

- *zmienna sterująca* oznacza zmienną sterującą pętlą. Zaleca się aby była o zmienna o typie całkowitym.
- *wartość początkowa* jest wartością, jaką przyjmuje zmienna sterująca przed rozpoczęciem wykonania instrukcji cyklu. Zaleca się aby była o zmienna o typie całkowitym.
- *wartość końcowa* jest wartością ograniczającą wykonanie kolejnego cyklu: realizacja pętli jest zakończona, gdy wartość zmiennej sterującej ją przekroczy. Zaleca się aby była o zmienna o typie całkowitym.

- `krok` jest parametrem opcjonalnym; jego brak jest równoważny przyjęciu wartości `krok=1`. Przy każdym obiegu pętli wartość zmiennej sterującej jest modyfikowana poprzez dodanie do niej wartości zmiennej `krok`. Zaleca się aby była o zmienna o typie całkowitym.

Liczba powtórzeń instrukcji cyklu wyraża się wzorem:

```
int ((wartość końcowa - wartość początkowa+krok) / krok )
```

Jeśli wynikająca z tego wzoru liczba powtórzeń jest równa bądź mniejsza od 0 to instrukcja cyklu nie zostanie wykonana ani razu.

Trzy przykładowe sytuacje z początku tego rozdziału zostaną teraz zapisane za pomocą instrukcji DO-indeksowane:

```
! sytuacja 1
DO i=0,9,1 ! albo równoważnie: DO i=0,9
    ....
    ....
END DO
```

Liczba powtórzeń: $(9 - 0 + 1) / 1 = 10$

albo:

```
! sytuacja 2
DO i=5, 50, 5
    ....
    ....
END DO
```

Liczba powtórzeń: $(50-5+5) / 5 = 10$

albo:

```
! sytuacja 3
DO i=10, 1, -1
    ....
    ....
ENDDO
```

Liczba powtórzeń: $(1-10+(-1)) / (-1) = 10$

Ostatni przykład pokazuje, że `krok` może mieć wartość ujemną.