8. Instrukcje sterujące

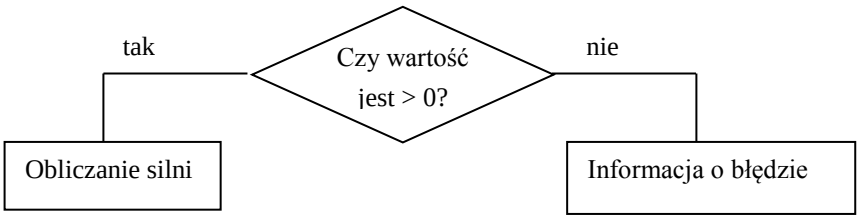
8.1. Instrukcja warunkowa

Instrukcja warunkowa pozwala na wykonanie polecenia lub zbioru poleceń pod pewnymi warunkami. Jeżeli warunki te są spełnione polecenia są wykonywane, w przeciwnym przypadku nie są wykonywane.

Przykład 1.

Należy napisać funkcję, która powinna obliczać silnię podanej przez użytkownika liczby. Funkcja taka miałaby więc jeden parametr wejściowy (liczbę której silnia powinna być obliczana) i jeden parametr wyjściowy (obliczona wartość silni). Dodatkowo funkcja powinna wyświetlać komunikat o błędzie, w przypadku, gdy podana przez użytkownika liczba nie jest liczbą dodatnią – w takim przypadku nie można obliczyć silni.

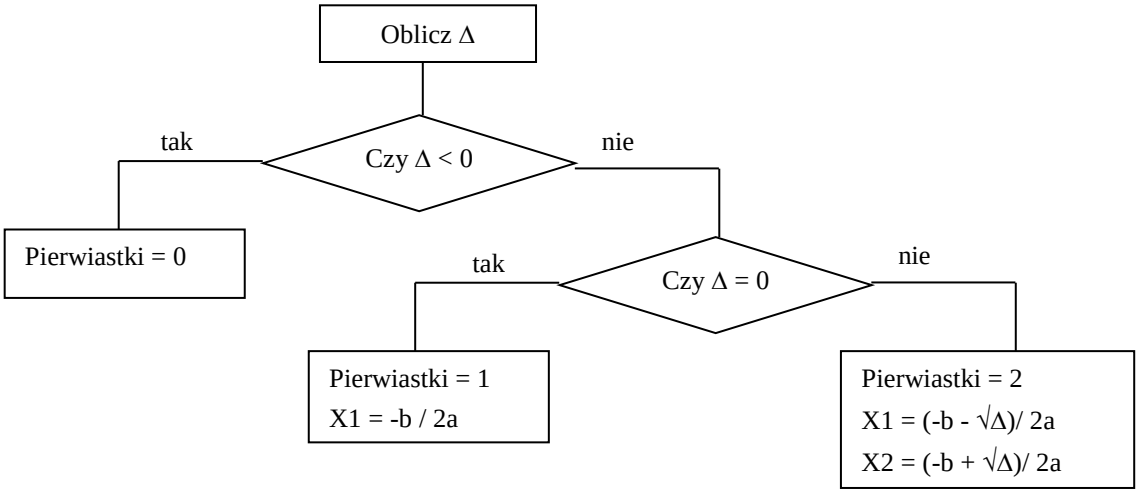
Podsumowując, funkcja powinna na początku sprawdzać, czy podana wartość wejściowa jest dodatnia, jeżeli tak: powinna wykonać odpowiednie obliczenia, jeżeli nie: powinna informować o błędzie.



Rys. 1. Schemat blokowy do przykładu 1

Przykład 2.

Należy napisać funkcję, która powinna obliczać pierwiastki równania kwadratowego. Funkcja powinna otrzymywać trzy parametry wejściowe (współczynniki równania kwadratowego: a, b oraz c). Funkcja powinna zwracać dwie wartości wyjściowe: w pierwszym parametrze należy zwracać liczbę pierwiastków rzeczywistych (wartość **0** – jeżeli Δ<0; wartość **1** – jeżeli Δ=0 i wartość **2** – jeżeli Δ>0), w drugim parametrze (który jest wektorem) należy zwracać obliczone pierwiastki.



Rys. 2. Schemat blokowy do przykładu 2

8.1.1. Wykorzystanie instrukcji warunkowej

Instrukcja warunkowa może się okazać bardzo przydatna również przy sprawdzaniu sposobu wywołania funkcji przez użytkownika. Większość błędów jest wychwytywana przez sam system.

Jeżeli przykładowa funkcja wymaga dwóch parametrów wejściowych i dwóch parametrów wyjściowych to system zgłasza błąd przy wywołaniu np.:

– z jednym parametrem wejściowym,

– z trzema parametrami wejściowymi,

– z trzema parametrami wyjściowymi.

System nie zgłasza błędów przy wywołaniu:

– z jednym parametrem wyjściowym,

– bez parametrów wyjściowych.

Błędy odnoszące się do zbyt małej ilości parametrów wejściowych podnoszone są dopiero w momencie próby wykorzystania przez funkcję parametru, któremu użytkownik nie nadał wartości w trakcie wywołania, tzn.:

można napisać np. funkcję, która w momencie braku wartości określonego parametru, przyjmuje domyślnie określoną wartość dla tego parametru.

Zestaw funkcji MATLAB-a, które mogą się okazać przydatne przy sprawdzaniu poprawności wywołania funkcji:

Funkcja	Opis
nargin	zwraca liczbę parametrów wejściowych, podatnych przez użytkownika podczas wywołania funkcji
nargout	zwraca liczbę parametrów wyjściowych, podatnych przez użytkownika podczas wywołania funkcji
disp	wyświetla w oknie **Command Window** tekst który jest je parametrem wejściowym, np.: >> **disp('Ala ma kota')** **Ala ma kota**
error	wyświetla w oknie **Command Window** tekst który jest jej parametrem wejściowym (tekst wyświetlany jest w kolorze czerwonym). Dodatkowo funkcja ta powoduje przerwanie działania funkcji wewnątrz której została użyta. >> **error('Ala ma kota')** **Ala ma kota**

return	Funkcja powoduje przerwanie działania funkcji wewnątrz której została użyta.
keyboard	Funkcja przerywa czasowo wykonanie funkcji wewnątrz której została użyta, umożliwiając użytkownikowi wpisywanie poleceń (np. nadających poprawne wartości parametrom wejściowym). Wpisywanie poleceń użytkownik musi zakończyć wpisując polecenie **return**. Dopiero po akceptacji tego polecenia wznawiane jest wykonanie zatrzymanej funkcji.

Przykład 3:

Należy napisać funkcję, która powinna sumować dwie macierze, które są parametrami wejściowymi funkcji. Funkcja powinna otrzymywać cztery parametry wejściowe (dwie macierze **A** i **B** oraz dwa dodatkowe współczynniki: **a**, **b**). Funkcja powinna zwracać wynikową macierz **C**. Macierz **C** należy obliczać jako: **C** = **a*****A** + **b*****B**. Jeżeli użytkownik nie poda wartości współczynników **a** i **b** należy przyjąć że są one równe 1.

8.1.2. Składnia instrukcji warunkowej

W zależności od potrzeb można wykorzystać uproszczoną postać instrukcji warunkowej:

(jeżeli istnieje potrzeba zareagowania tylko w jeden sposób):

```
if wyrażenie_warunkowe
  ciąg_instrukcji
end
```

lub (jeżeli istnieje potrzeba zareagowania na dwa sposoby – patrz przykład 1):

```
if wyrażenie_warunkowe
  ciąg_instrukcji_1
else
  ciąg_instrukcji_2
end
```

czy postać rozbudowaną (przykład 2):

if *wyrażenie_warunkowe1*

ciąg_instrukcji_1

elseif *wyrażenie warunkowe2*

ciąg_instrukcji_2

...

else

ciąg_instrukcji_N

end

Ciąg instrukcji wykonywany jest gdy *wyrażenie_warunkowe* ma wartość prawda (macierz o wszystkich elementach różnych od zera). Bloki **elseif** i **else** są opcjonalne.

8.1.3. Wyrażenia warunkowe

Definiując wyrażania warunkowe można wykorzystywać **operatory relacyjne**:

Operator	Znaczenie	Przykład	Opis
>	„większy niż”	**x > 0**	Czy **x** jest większe od zera?
		y > 5	Czy **y** jest większe od pięciu?
		z > x	Czy **z** jest większe od **x**?
>=	„większy lub równy”	**x >= 0**	Czy **x** jest większe lub równe zero?
		y >= 5	Czy **y** jest większe lub równe pięć?
		z >= x	Czy **z** jest większe lub równe **x**?
<	„mniejszy niż”	**x < 0**	Czy **x** jest mniejsze od zera?
		y < 5	Czy **y** jest mniejsze od pięciu?
		z < x	Czy **z** jest mniejsze od **x**?

<=	„mniejszy lub równy”	**x <= 0** **y <= 5** **z <= x**	Czy **x** jest mniejsze lub równe zero? Czy **y** jest mniejsze lub równe pięć? Czy **z** jest mniejsze lub równe **x**?
== (dwa znaki równości)	„równy”	**x == 0** **y == 5** **z == x**	Czy **x** jest równe zero? Czy **y** jest równe pięć? Czy **z** jest równe **x**?
~=	„różne”	**x ~= 0** **y ~= 5** **z ~= x**	Czy **x** jest różne od zera? Czy **y** jest różne od pięciu? Czy **z** jest różne od **x**?

oraz **operatory logiczne**:

Operator	Znaczenie	Przykład	Opis
&	oraz, i	**(x > 0) & (x <= 10)**	Czy **x** $\in (0, 10]$?
| (kreska pionowa)	lub	**(x < 0) | (x >= 10)**	Czy **x** $\in (-\infty, 0) \cup [10, +\infty]$?
~	nie	**~(x>0)**	Czy **x** ≤ 0?

Przykład 1 cd.

Zadanie można wykonać na wiele różnych sposobów. Poniżej podano dwie różne wersje.

Wersja 1	Wersja 2
function [s]=Silnia(n) **if n > 0** ** s = prod(1 : n);** **else** ** error('Parametr wejśc. powinien być > 0');** **end**	**function [s]=Silnia(n)** **if n > 0** ** s = prod(1 : n);** **end** **if n <= 0** ** error('Parametr wejśc. powinien być > 0');** **end**

Przykład 2. cd.

Podobnie jak w przypadku przykładu poprzedniego, zadanie można wykonać na wiele różnych sposobów:

Wersja 1	Wersja 2
function [pierwiastki, x]=RKwadrat(a, b, c) **delta = b^2 – 4*a*c;** **if delta < 0** ** pierwiastki = 0;** **elseif delta == 0** ** pierwiastki = 1;** ** x = -b / (2*a);** **else** ** pierwiastki = 2;** ** x = [(-b–delta^0.5)/(2*a), ...** ** (-b+delta^0.5)/(2*a)];** **end**	**function [pierwiastki, x]=RKwadrat(a, b, c)** **delta = b^2 – 4*a*c;** **if delta < 0** ** pierwiastki = 0;** **end** **if delta == 0** ** pierwiastki = 1;** ** x = -b / (2*a);** **end** **if delta >= 0** ** pierwiastki = 2;** ** x(1) = (-b – delta^0.5) / (2*a) ;** ** x(2) = (-b + delta^0.5) / (2*a);** **end**

Przykład 3. cd.

Podobnie jak w przypadku przykładów poprzednich, zadanie można wykonać na wiele różnych sposobów:

Wersja 1	Wersja 2
function [C]=Suma(A, B, a, b) **if nargin == 2** **a = 1;** **b = 1;** **end** **if nargin == 3** **b = 1;** **end** **C = a*A + b*B;**	**function [C]=Suma(A, B, a, b)** **if nargin < 4** **if nargin < 3** **a = 1;** **end** **b = 1;** **end** **C = a*A + b*B;**

Dodatkowo funkcja mogłaby sprawdzać np. czy macierze **A** i **B** mogą być do siebie dodane (czy mają takie same wymiary).

Wersja 3

```
function [C]=Suma(A, B, a, b)

if nargin == 2
  a = 1;
  b = 1;
end
if nargin == 3
  b = 1;
end
if ( (size(A,1) ~= size(B, 1)) | (size(A,2) ~= size(B, 2)) )
  error('Macierze wejściowe mają niezgodne wymiary');
else
  C = a*A + b*B;
end
```

8.2. Instrukcja iteracyjna for

Instrukcja iteracyjna **for** pozwala na wielokrotne wykonanie polecenia lub zbioru poleceń dla kolejnych wartości zmiennej iterowanej.

Przykład 4.

Należy napisać funkcję, która wypełnia wektor wyjściowy kwadratami kolejnych liczb naturalnych. Funkcja otrzymuje w postaci parametru wejściowego rozmiar wektora wyjściowego (tzn.: ilość elementów do wypełnienia).

Funkcja mogłaby wyglądać np. następująco:

```
function [y] = Ustaw(n)


x = 1:n;
y = x.^2;
```

Załóżmy, że w rozwiązaniu zadania nie można wykorzystać operacji y = x.^2.

8.2.1. Składnia instrukcji iteracyjnej for

for zmienna_iterowana=macierz_wartości

ciąg_instrukcji

end

Działanie instrukcji polega na wykonaniu ciągu instrukcji dla kolejnych wartości zmiennej iterowanej. Wartościami zmiennej iterowanej są kolejne wektory kolumnowe z macierzy wartości.

Rozważmy ciąg instrukcji:

Przykład	Wygląd ekranu po wykonaniu instrukcji	Opis

`A=[1 2 3; 4 5 6];`
`for i=A`
`  i`
`end`
	`i =`
	`  1`
	`  4`
	
	`i =`
	`  2`
	`  5`
	
	`i =`
	`  3`
	`  6`
		A jest macierzą: $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$. Zgodnie z opisem składni instrukcji **for** zmienna iterowana **i** będzie otrzymywała wartości równe kolejnym kolumnom macierzy **A**, tzn.: $\begin{bmatrix}1\\4\end{bmatrix}$, $\begin{bmatrix}2\\5\end{bmatrix}$ oraz $\begin{bmatrix}3\\6\end{bmatrix}$. Dla każdej wartości zmiennej iterowanej będzie wykonywana instrukcja **i**. Wykonanie tej instrukcji każdorazowo spowoduje wyświetlenie aktualnej wartości zmiennej iterowanej.

Przykład 4. cd.

<table>
<tr><th>Wersja 1</th><th>Opis</th></tr>
<tr><td>function [y]=Ustaw(n)

x = 1:n;
for i=x
x(i)=i^2;
end</td><td rowspan="4">Załóżmy, że n ma wartość 3. Stąd zmienna iterowana i dostaje kolejno wartości: 1, 2, 3 (macierz wartości jest równa: [1 2 3], stąd kolejne kolumny tej macierzy zawierają wartości: 1, 2 oraz 3).
Na początku zmienna i ma wartość 1
wykonywana jest instrukcja x(1) = 1^2 (która powoduje przypisanie pierwszemu elementowi wektora x wartości 1)
Następnie zmienna i otrzymuje wartość 2
wykonywana jest instrukcja x(2) = 2^2 (która powoduje przypisanie drugiemu elementowi wektora x wartości 4)
Na koniec zmienna i otrzymuje wartość 3
wykonywana jest instrukcja x(3) = 3^2 (która powoduje przypisanie trzeciemu elementowi wektora x wartości 9)</td></tr>
<tr><td>Wersja 2</td></tr>
<tr><td>function [y]=Ustaw(n)

for i=1:n
x(i)=i^2;
end</td></tr>
</table>

Uwaga!

Przykładowe zadanie 4 można było rozwiązać bez wykorzystywania pętli (takie rozwiązanie podano pod sformułowaniem zadania). Istnieją jednak problemy, których nie można rozwiązać bez wykorzystania pętli.

8.3. Instrukcja iteracyjna while

Instrukcja iteracyjna **while** pozwala na wielokrotne wykonanie polecenia lub zbioru poleceń pod warunkiem spełnienia wyrażenia warunkowego. Jeżeli to wyrażenie ma wartość logiczną „prawda” polecenia są wykonywane, jeżeli wyrażenie ma wartość logiczną „fałsz” pętla kończy swoje działanie. Pętla **while** pozwala więc na wielokrotne wykonanie ciągu poleceń, przy czym liczba przebiegów pętli nie jest z góry określona, zależy od wyrażenia warunkowego.

Składnia instrukcji iteracyjnej while

while wyrażenie_warunkowe

ciąg_instrukcji

end

Przykład 5.

Należy napisać funkcję, która wypełnia wektor wyjściowy kolejnymi elementami ciągu geometrycznego. Parametrami wejściowymi funkcji są: pierwszy element ciągu: **a1**, iloraz ciągu: **q**, oraz wartość maksymalnego elementu ciągu **an**. Element ciągu należy generować tak długo aż wartość kolejnego elementu przekroczy zadaną wartość **an**.

Funkcja	Opis
function [x]=Ustaw(a1, q, an) **i = 1;** **ai = a1;** **while ai <= an** **x(i)=ai;** **i = i +1;** **ai = ai*q;** **end**	Załóżmy, że funkcja została wywołana z parametrami: **a1** = 1; **q** = 4; **an** = 10. Zmienna pomocnicza **i** została wprowadzona, aby przechowywać indeks aktualnie generowanego elementu wektora, zmienna **ai** przechowuje wartość tego elementu. Przed wykonaniem pętli zmienne otrzymują wartości początkowe: **i**=1 oraz **ai** = **a1**. Jeżeli obliczone **ai** jest mniejsze od zadanego maksymalnego elementu **an**: element ten trafia na pozycję **i** do wektora wyjściowego **x**. Dla przykładowego ciągu nastąpi przypisanie: **x**(1) = 1. Kolejne dwa polecenia mają na celu przesunięcie o 1 wartości zmiennej **i** oraz obliczenie wartości kolejnego elementu ciągu: **i** otrzyma wartość 2, **ai** wartość 4. Następuje ponowne sprawdzenie czy nowy element ciągu, nie przekracza zadanej wartości maksymalnej oraz przypisanie : **x**(2) = 4. Następnie modyfikowane są **i** (otrzymuje wartość 3) oraz **ai** (otrzymuje wartość 16). Następuje ostatnie już sprawdzenie czy element **ai** nie przekracza wartości **an**. Ze względu na to, że 16 > 10, pętla kończy swoje działanie. Ostatecznie wektor **x** zawiera tylko dwa elementy: 1, 4.

Ćwiczenia

1. Zmodyfikuj funkcję „Kolo” (zadanie 4, temat: „Funkcje”), tak aby mogła otrzymywać dodatkowy parametr (1 lub 2) określający która wielkość ma być obliczona (1 – pole, 2 – obwód). Jeżeli parametr ten nie będzie podany funkcja powinna działać tak jak w zadaniu 4.
2. Zmodyfikuj funkcję „Prostopadloscian” (zadanie 5, temat: „Funkcje”), tak aby sprawdzała czy parametry wejściowe są większe od zera i generowała błąd jeżeli warunek ten nie jest spełniony.
3. Zmodyfikuj funkcję „UkladRownan” (zadanie 7, temat: „Funkcje”), tak aby sprawdzała dodatkowo czy wartość wyznacznika macierzy współczynników jest różna od zera i generowała błąd jeżeli warunek ten nie jest spełniony.
4. Zmodyfikuj funkcję „UkladRownan” (zadanie 3), tak aby sprawdzała:

   – czy macierz wejściowa jest macierzą kwadratową,

   – czy wymiar wektora wejściowego odpowiada ilości wierszy i kolumn macierzy.

   Jeżeli warunki te nie są spełnione należy wyświetlać odpowiednie komunikaty i przerwać działanie funkcji.

   Dodatkowo jeżeli wektor wejściowy ma właściwy rozmiar, ale jest wektorem wierszowym, należy wykonać transpozycję tego wektora.

Wskazówka

Aby w prosty sposób uwzględnić wszystkie wymienione warunki – można postąpić następująco:

1. Jeżeli macierz (będąca pierwszym parametrem funkcji) nie jest macierzą kwadratową – wygenerować błąd (np. „Macierz musi być kwadratowa”).
2. Jeżeli drugi parametr funkcji nie jest wektorem (ani wierszowym ani kolumnowym) – wygenerować błąd (np. „Drugi parametr musi być wektorem”).
3. Jeżeli wektor wejściowy (drugi parametr funkcji) jest wektorem wierszowym – wykonać transpozycję.
4. Jeżeli liczba kolumn macierzy jest różna od ilości wierszy wektora – wygenerować błąd (np. „Macierz i wektor mają niezgodne wymiary”).

Jeżeli dla parametrów wejściowych nie zostaną podniesione błędy (omówione w punktach: 1., 2. i 4.) oznacza to, że zadanie może być rozwiązywane dalej (należy uwzględnić modyfikacje wprowadzane w zadaniu 3).