5. Programowanie

5.1 Instrukcje sterujące

Poza opisanymi powyżej komendami systemowymi oraz funkcjami i operacjami matematycznymi i tekstowymi, interpretery poleceń w Matlabie i Scilabie obsługują jeszcze typowe instrukcje sterujące działaniem programów: instrukcje warunkowe if i swich oraz pętle for i while.

W poniższym zestawieniu instrukcji sterujących Matlaba i Scilaba wyróżniono kolorem różnice w składni (Scilab oferuje różne warianty składni, co wynika z konieczności zachowania zgodności z Matlabem i ze swoimi starszymi wersjami). Najpoważniejsze różnice to oznaczenie komentarza (// lub %) i instrukcja switch (słowo kluczowe select lub switch).

// komentarz (do końca linii)		% komentarz (do końca linii)

if *warunek1*
  kod1
[**elseif** *warunek2*
  kod2]
[**else**
  kod3]
end
	if *warunek1* **then**
	  kod1
	[**elseif** *warunek2* **then**
	  kod2]
	[**else**
	  kod3]
	end
		if *warunek1*
		  kod1
		[**elseif** *warunek2*
		  kod2]
		[**else**
		  kod3]
		end

select *wyrazenie*
case *wartosc1*
  kod1
case *wartosc2*
  kod2
[**else**
  kod3]
end
	select *wyrazenie*
	case *wartosc1* **then**
	  kod1
	case *wartosc2* **then**
	  kod2
	[**else**
	  kod3]
	end
		switch *wyrazenie*
		case *wartosc1* **then**
		  kod1
		case *wartosc2* **then**
		  kod2
		[**otherwise**
		  kod3]
		end

for *i=pocz:krok:koniec*
  kod
end
	for *i=pocz:krok:koniec* **do**
	  kod
	end
		for *i=pocz:krok:koniec*
		  kod
		end

while *warunek*
  kod1
[**else**
  kod2]
end
	while *warunek* **do**
	  kod1
	[**else**
	  kod2]
	end
		while *warunek*
		  kod1
		end

	while *warunek* **then**
	  kod1
	[**else**
	  kod2]
	end	

Zaleca się na zakończenie każdej funkcji kodu wstawiać znak średnika. Brak średnika oznacza włączenie echa, czyli wyświetlanie odpowiedzi na konsoli komend, co pomaga śledzić przebieg realizacji funkcji ale spowalnia obliczenia (w Scilabie może dodatkowo wstrzymywać realizację funkcji ponieważ czasem trzeba potwierdzić coś naciskając klawisz).

Instrukcje, podobnie jak inne polecenia i funkcje, mogą być zapisywane w jednej linii, a do oddzielenia poszczególnych części stosuje się przecinek albo średnik, przy czym średnik jednocześnie wyłącza echo. Zapis złożonych instrukcji w jednej linii umożliwia ich zastosowanie nie tylko w skryptach i funkcjach ale także w linii komend. Na przykład:

if *warunek*, *kod1*, **else** *kod2*, **end** **if** *warunek* **then** *kod1*, **else** *kod2*, **end**	**if** *warunek*, *kod1*, **else** *kod2*, **end**
select *wyrazenie*, **case** *wartosc1*, *kod1*, **else** *kod2*, **end**	**switch** *wyrazenie*, **case** *wartosc1*, *kod1*, **otherwise** *kod2*, **end**
for *i=pocz:krok:koniec*, *kod*, **end** **for** *i=pocz:krok:koniec* **do** *kod*, **end**	**for** *i=pocz:krok:koniec*, *kod*, **end**
while *warunek*, *kod1* [,**else** *kod2*], **end** **while** *warunek* **do** *kod1* [,**else** *kod2*], **end**	**while** *warunek*, *kod1*, **end**

Można również dzielić długie polecenia - wstawiając na końcu linii trzy kropki po spacji (**...**) i wówczas następna linia będzie interpretowana jako ciąg dalszy polecenia.

5.2 Skrypty

Skrypt to zewnętrzny plik tekstowy zawierający sekwencje instrukcji i poleceń, które mogą być zrealizowane w przestrzeni roboczej (Workspace) Matlaba/Scilaba, to znaczy są uruchamiane w środowisku Matlaba/Scilaba i działają na zmiennych w tym środowisku. Uruchomienie wykonuje się zwykle pod edytorem tekstowym lub z konsoli Matlaba/Scilaba (p. 2.2.2). Działanie skryptu jest takie samo jak działanie analogicznej sekwencji poleceń wykonywanej w linii komend – można więc również zawartość skryptu skopiować do konsoli komend i tam wykonać, również wówczas gdy skrypt zawiera instrukcje zapisane w kilku liniach, ponieważ uruchomienie skopiowanej sekwencji następuje dopiero po naciśnięciu klawisza Enter.

Zastosowanie instrukcji sterujących pozwala na zautomatyzowanie realizacji programu badań. Warto je stosować nawet do realizacji prostych zadań jako pewną formę dokumentowania przeprowadzonego badania, które można poza tym zawsze powtórzyć.

Skrypty mogą zawierać wszystkie rodzaje poleceń (komendy systemowe, funkcje i operacje matematyczne i tekstowe) oraz odwołania do innych skryptów i funkcji zdefiniowanych przez użytkownika. Skrypty mogą odwoływać się do wszystkich zmiennych w przestrzeni roboczej a także definiować nowe zmienne w tej przestrzeni, które w niej pozostają po zakończeniu działania skryptu. Przekazywanie informacji pomiędzy skryptami następuje poprzez takie zmienne, ewentualnie za pośrednictwem różnego typu plików (skrypty nie zwracają wartości).

5.3 Funkcje użytkownika

Funkcja to sekwencja instrukcji i poleceń, zwykle zapisana w zewnętrznym pliku, która może mieć zdefiniowane parametry wejściowe (argumenty) i która może zwracać wartości parametrów wyjściowych (wynik).

5.3.1 Funkcje definiowane z linii komend

Jednym ze sposobów definiowania prostych funkcji użytkownika jest zastosowanie funkcji definiującej, w której podawana nazwa, wzór i parametry definiowanej funkcji

deff('[*wy*]=*nazwa*(*arg1*, *arg2*, ...)', '*wy=wzor*')	*nazwa* = **inline**('*wzor*', '*arg1*', '*arg2*, ...')

Przykład funkcji definiowanych bezpośrednio z linii komend: 1) z=sin(x)*cos(x), 2) $y=x_1+x_2$

- definicja 1: **deff**('y=z(x)', 'y=cos(x)+sin(x)', 'x') - definicja 2: **deff**(' [y]=mojplus(x1,x2)', 'y=x1+x2') - wywołania: z(%pi), mojplus(1,2) → -1 3	- definicja 1: z=**inline**('cos(x)+sin(x)', 'x'); - definicja 2: mojplus = **inline**('x1+x2', 'x1', 'x2') - wywołania: z(pi), mojplus(1,2) → -1 3

Funkcje zdefiniowane w ten sposób są przechowywane w przestrzeni roboczej do czasu zakończenia pracy z programem.

5.3.2 Funkcje definiowane w pliku

Funkcje definiowane przez użytkownika w pliku tekstowym muszą zachować strukturę, która pozwala określić nazwę funkcji oraz jej parametry wejściowe i wyjściowe (to główna cecha, na podstawie której programy odróżniają funkcję od skryptu[1]). W typowej definicji funkcja jest zawarta w pliku o tej samej nazwie:

Nazwa pliku: *nazwafun*.sci

```
function[wy1, ...] = nazwafun(arg1,...)
  instrukcje
  wy1=...                    – podstawienie wartości wyjściowej
endfunction
```

Nazwa pliku: *nazwafun*.m

```
function[wy1, ...] = nazwafun(arg1,...)
  instrukcje
  wy1=...                    – podstawienie wartości wyjściowej
[end]                        – niekonieczne jeśli jedna funkcja
```

Jako przykład takiej definicji przedstawiono zawartość pliku o nazwie 'iloczyn' (obliczenie iloczynu liczb naturalnych od 1 do n):

```
function[a] = iloczyn(n)
    a=1
    for i = 1:n;  a = a*i;  end
endfunction
```

```
function[a] = iloczyn(n)
    a=1
    for i = 1:n;  a = a*i;  end
end
```

Wywołanie funkcji zapamiętanej w pliku 'iloczyn' ma postać:

– wczytanie funkcji z pliku do przestrzeni roboczej:
  exec(„iloczyn.sci");
– użycie funkcji (wywołanie funkcji z przestrzeni roboczej):
  x = iloczyn(4)

– plik z funkcją musi być w kartotece bieżącej lub zdefiniowanej w ścieżkach przeszukiwania (główna konsola: File/Set Path)
– użycie funkcji
  x = iloczyn(4)

W kolejnym przykładzie plik 'modulo10' zawiera definicje dwóch funkcji:

```
function [ile, reszta, pelne]=modulo10(n)
   reszta = modulo(n,10);
   ile = (n-reszta)/10;
   pelne = pelne10(n);
endfunction

function [p]=pelne10(n)
   if modulo(n,10)> 0; p=0; else p=1; end
endfunction
```

```
function[ile, reszta, pelne] = modulo10(n)
   reszta = mod(n,10);
   ile = (n-reszta)/10;
   pelne = pelne10(n);
end

function[p] = pelne10(n)
   if mod(n,10)> 0; p=0; else p=1; end
end
```

Główna funkcja modulo10 zwraca tym razem trzy parametry i używa funkcji pomocniczej pelne:

```
exec('modulo10.sci');        – wczytanie funkcji z pliku
[x r dzies]  = modulo10(12)
dzies2 = pelne10(14)         – wykorzystanie f.pomocniczej
```

```
[x r dzies]  = modulo10(12)
%dzies2 = pelne10(12)        - f.pomocnicza niedostępna
```

W przypadku Scilaba nazwy funkcji nie muszą być takie same jak nazwy plików i wszystkie funkcje w zdefiniowane pliku są dostępne (także funkcje pomocnicze), ponieważ program nie poszukuje funkcji w plikach ale w przestrzeni roboczej. Informację o wczytanych funkcjach można uzyskać za pomocą komendy whos[2].

[1] Scilab przewiduje rozszerzenie *.sce dla pliku skryptu i *.sci dla pliku funkcji, ale ma to charakter porządkowy bo nazwę pliku podaje się zawsze z rozszerzeniem
[2] komenda wyświetla wszystkie nazwy występujące w przestrzeni roboczej (zmienne, funkcje, biblioteki, ...)

W funkcji można operować na parametrach wejściowych i wyjściowych ale tylko wartości parametrów wyjściowych zostaną przekazane do miejsca wywołania funkcji. Zmienne zainicjowane we funkcji są zmiennymi lokalnymi i znikają po wyjściu z funkcji. Poza tym w funkcji można zainicjować i odwoływać się do zmiennych globalnych. W Scilab można też odczytać wartości zmiennych z przestrzeni roboczej ale w momencie zapisania wartości tworzony jest lokalny duplikat zmiennej, który znika po wyjściu z funkcji (zmiany nie zostaną przekazane do miejsca wywołania). Funkcje są bardziej czytelne jeśli zamiast odwoływania się do zmiennych z przestrzeni roboczej czy zmiennych globalnych ich wartości zostaną przekazane poprzez listę parametrów wejściowych.

Poza typowymi instrukcjami i poleceniami, wewnątrz funkcji stosuje się także polecenie wyjścia z funkcji w określonym miejscu oraz odczytanie ilości parametrów wejściowych i wyjściowych:

return	return
[lsh rhs] = argn()	nargin, nargout

5.4 Zasięg zmiennych

Podstawowy obszar pamięci, na którym operują Matlab/Scilab nazywany jest przestrzenią roboczą (Workspace). W przestrzeni roboczej istnieją zmienne, które powstają przez realizację poleceń w linii komend lub w skrypcie. Zmienne z przestrzeni roboczej można:

odczytać – w linii komend, w skryptach, w funkcjach[1]	odczytać – w linii komend, w skryptach
zapisać - w linii komend, w skryptach	zapisać - w linii komend, w skryptach

Poza zmiennymi w podstawowej przestrzeni roboczej programu można definiować zmienne globalne, które są dostępne do odczytu i zapisu we wszystkich obszarach (skryptach, funkcjach), gdzie zostaną zadeklarowane jako globalne. Dla zachowania przejrzystości skryptów i funkcji powinno to być rozwiązanie stosowane z dużym umiarem.

Zmienne lokalne to dostępne tylko w obszarze funkcji, w której zostały zdefiniowane – zanikają po wyjściu z funkcji.

[1] lepiej unikać tego rozwiązania i przekazywać wartość przez listę parametrw funcji