3. Въвеждане и извеждане на информация. Оператори за управление на изчислителния процес.

Всички оператори и функции могат да се използват в директен режим на работа или в програмен режим в състава на програма за обработка на данни. При стартиране на създадена програма системата изпълнява последователно всички команди от началото до края на файла.

3.1. Въвеждане на информация

Въвеждането на данни в диалогов режим се извършва с помощта на командата **input**. Използват се два основни варианта:

- *var* = **input**('*String*') – въвеждане на числени данни;
- *var* = **input**('*String*','*s*') – въвеждане на низ.

където:

var - име на променливата за която се въвежда стойност. Тя може да бъде скалар, вектор, матрица, низ;

String - произволен текст, който се извежда на екрана при изпълнението на командата. Този текст подсеща/дава инструкции какви данни да се въведат, например 'Въведете стойност за коефициента a: '. Възможно е текста да се изведе на няколко реда. Пренасянето на нов ред става с помощта на символа \n;

s - ключ за въвеждане на низ. При задаване на този ключ не е необходимо въвеждания низ да се загражда в апострофи.

Командата **input** се използва в програмен режим. При изпълнение на програма, която съдържа команда **input**, когато системата срещне тази команда, тя спира изпълнението на програмата в очакване да бъдат въведени съответните данни. На екрана се изписва зададения чрез *String* текст и след този текст се въвеждат данните. Въвеждането завършва с натискане на клавиша Enter. Системата присвоява въведената стойност на променливата *var* и продължава изпълнението на програмата. Команда **input** е част от почти всички примерни програми, които се разглеждат в този материал.

Примери за използване на команда input:

```
a=input('въведете стойност за a ')
въведете стойност за a [1 1 1
2 2 2
3 3 3] Enter
```

Полученият резултат от действието на зададената команда означава следното: системата извежда на екрана текста 'въведете стойност за a ', така като е зададено в командата; потребителя въвежда стойности за елементите на матрицата според правилата, в случая

```
[1 1 1
2 2 2
3 3 3];  % натиска се клавиш Enter. Полученият резултат е:
```

```
a =
     1     1     1
     2     2     2
     3     3     3
```

```
a=input('въведете стойност за a ','s')
въведете стойност за a abc Enter
```

При този пример за променива a се въвежда символна стойност 'abc'.

```
a =
     abc
```

3.2. Извеждане на информация

Командата **disp** е предназначена за извеждане на данни (числени и символни) и текст. Характерна особеност на тази команда е, че тя приема само един аргумент. Ако трябва да се извеждат на един ред няколко числа заедно с пояснителен текст, те трябва да се обединят в един обект с помощта на конкатенация (обединяване). При това числата трябва да се преобразуват в низове с помощта на функцията **num2str**. Ако липсва текст, числата трябва да се представят просто като елементи на един вектор.

Примери:

```
>> disp (a)
     abc

>> b=2;
>> disp(b)
     2

>> disp([' b= ',num2str(b)])
b= 2

>> c=3;
>> d=4;
>> disp([b c d])
     2     3     4

>> disp(['b =', num2str(b),' c =',num2str(c), ' d =',
num2str(d)])
b=2 c=3 d=4
```

Аналогично на **disp** работят и функциите **warning** и **error**:

- **warning**('*String*') – извежда предупредително съобщение и продължава изпълнението на програмата;
- **error**('*String*') – извежда съобщение за грешка и прекратява изпълнението на програмата.

В програмите се налага да се реализират разклонени и циклични алгоритми. Това е възможно като се използват управляващи оператори. Тези оператори дават възможност да се организират циклични пресмятания, да се изпълняват разклонения на програмата и при възникнали специфични условия да се прекратят пресмятанията или действието на програмата.

В MATLAB са включени управляващи оператори, подобни на тези, с които разполагат по-голямата част от съвременните езици за програмиране. Пълен списък на всички оператори и функции, които се използват за програмиране в MATLAB, се получава с командата

```
>>help lang.
```

Основните управляващи оператори са:

- **if** – оператор за условен преход;
- **switch** – превключващ оператор;
- **for** – аритметичен оператор за цикъл;
- **while** – условен (логически) оператор за цикъл;
- **try** – оператор за обработка на аварийни ситуации.

Като управляващи оператори се разглеждат и оператори за реализиране на следните специфични действия:

- **break, continue, pause, keyboard, return** - прекъсване, прескачане на цикъл, пауза, предаване управлението на клавиатурата, връщане в извикваща програма;
- **input, disp, warning, error** Въвеждане и извеждане на информация;
- **menu** - организиране на менюта.

Всички управляващи оператори в MATLAB имат вид на сложен оператор, който започва със служебната дума **if, switch, for, while** или **try** и завършват с думата **end**.

3.3. Оператор за условен преход `if`

Възможни са три варианта на този оператор.

Вариант 1:

```
if logexpr
    statements;
end
```

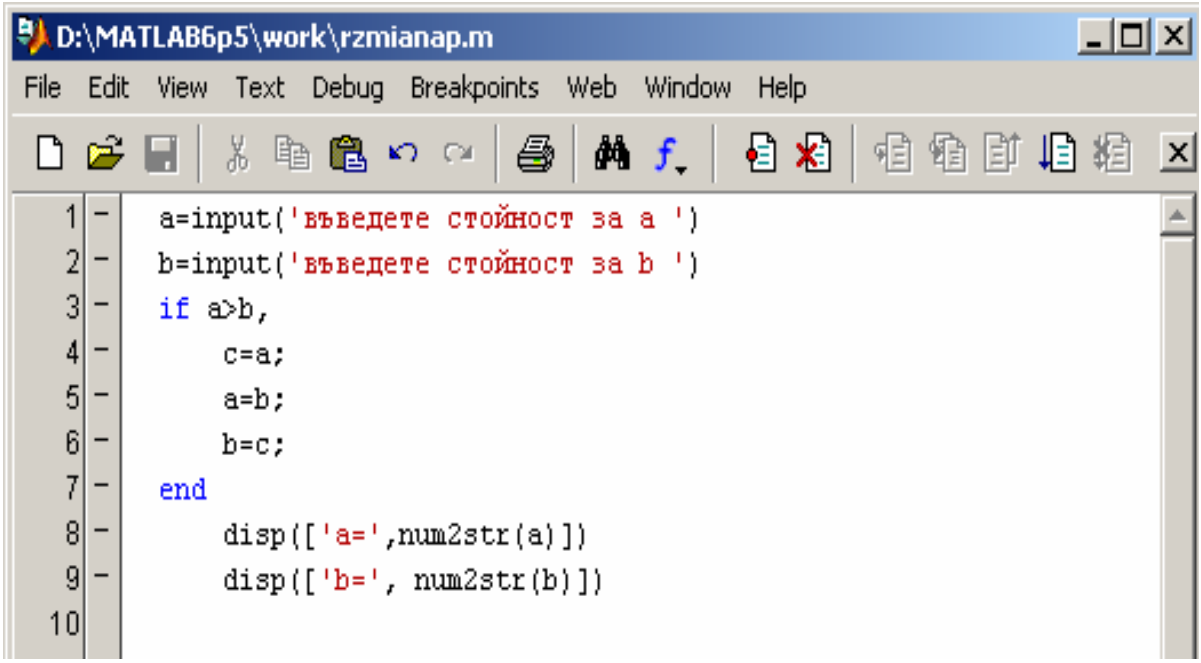
Действието на този оператор е следното: ако логическият израз *logexpr* има стойност True (истина), се изпълняват операторите *statements,* след което управлението се предава на оператора след **end**. Ако стойността на логическия израз *logexpr* е False (неистина), управлението се предава на оператора след **end**. Тази конструкция на оператора съответства на кратката форма на базовата алгоритмична структура разклонение (алтернатива).

Допустимо е разположението на тези оператори на един ред:

```
if logexpr, stat1; stat2; stat3; …; end
```

Пример: Да се състави програма за размяна на стойностите на две числа a и b, само ако е изпълнено условието a>b.

Примерът е разгледан в лекцията за разклонени алгоритми. За размяната на две стойности са необходими три оператора за присвояване. На Фиг.3.1. е представена програмата, написана като файл. Работа с файлове в MATLAB се разглежда в Тема 4.



Фиг.3.1 Програма за размяна на стойностите на две числа

Вариант 2:

```
if logexpr
    statements1;
else
    statements2;
end
```

Действието на този оператор е следното: ако *logexpr* има стойност True се изпълняват операторите *statements1*, след което управлението се предава на оператора след **end**. Ако стойността на *logexpr* е False се изпълняват операторите *statements2,* след което управлението се предава на оператора след **end**. Тази конструкция на оператора съответства на пълната форма на базовата алгоритмична структура разклонение (алтернатива).

Пример: Да се изчисли за различни стойности на аргумента x стойността на функцията $y=f(x)$, която е зададена по следния начин:

$$y = \begin{cases} 0, & \text{за } |x| > R \\ \sqrt{R^2 - x^2} & \text{за } |x| <= R \end{cases}$$

Този пример е разгледан в темата за разклонени алгоритми, където е представена и блок-схема на алгоритъма за решаване на задачата.

Програма:

```
R=input('въведи радиус R ')
x=input('въведи x ')
if abs(x)>R,
    y=0
else
    y=sqrt(R^2-x^2)
end
disp(['y=', num2str(y)])
```

Програмата е съхранена като файл **iffunp.m** и се изпълнява, като в командния ред се задава името на файла

```
>> iffunp
въведи радиус R 2

R =
     2

въведи x 4

x =
     4

y=0
```

```
>> iffunp
въведи радиус R 4

R =
     4

въведи x 2

x =
     2

y=3.4641
```

Вариант 3:

```
if logexpr1
    statements1;
```

```
elseif logexpr2
    statements2;
elseif logexpr3
    statements3;
..............
else
    statements;
end
```

Действието на оператора е следното: ако *logexpr1* е True, изпълняват се оператори *statements1*, след което управлението се предава на оператора след **end**. Ако е False, проверява се за истинност изразът *logexpr2*. Ако той има стойност True, тогава се изпълняват оператори *statements2*, след което управлението се предава на оператора след **end**. Ако стойността на *logexpr2* е False, проверява се *logexpr3* и т. н. Ако нито един от логическите изрази няма стойност True, тогава се изпълняват операторите *statements*. На практика с такава конструкция на условния оператор **if** се реализира многостранно разклонение и броят на **elseif** е произволен. Операторът **else** не е задължителен. Тази конструкция на оператора съответства на вложени една в друга базови алгоритмични структури разклонение (алтернатива).

В MATLAB**,** както и в други програмни среди**,** логическата константа True се изразява чрез числото 1 (единица), а логическата константа False – чрез 0 (нула).

3.4. Логически оператори и функции

За записване на логически изрази *logexpr* се използват оператори за сравнение (релации) и логически оператори.

3.4.1. Оператори за сравнение:

< − по-малко;
<= – по-малко или равно;
> – по-голямо;
>= – по-голямо или равно;
== – равно;
~= – не е равно.

Важно е да се запомни, че логическото равенство се представя с два последователни символа за равенство " == ". Един символ за равенство се използва само в операторите за присвояване.

3.4.2. Логически оператори:

& – поелементно логическо "И" (and);
&& – кратък вариант на логическо "И";
| – поелементно логическо "ИЛИ" (or);
|| – кратък вариант на логическо "ИЛИ";
~ – логическо "НЕ" (not).

Разликата в действието на поелементните и кратките варианти на логическото "И" е следната:

- `A & B` – сравняват се поелементно матриците A и B и резултатът е матрица, състояща се от логически единици 1 (True), там, където и двата елемента на A и B са различни от нула, и логически нули 0 (False) в останалите позиции. Двете матрици A и B трябва да са с еднаква размерност. Възможно е една от двете матрици да е скалар. В частност и двата аргумента A и B могат да бъдат скалари.
- `a && b` – сравняват се логическите скалари a и b и се връща 1 (True) ако и двата израза са истини или 0 (False) в противен случай. Ако a има стойност False, тогава b не се пресмята, тъй като стойността на целия логически израз (False) няма да зависи от b. От тук идва и названието "Кратко И".

По аналогичен начин действат и логическите оператори | и ||:

- `A | B` – връща се логическа матрица от 1 и 0. Единиците съответстват на тези двойки елементи на матриците A и B, при които поне единият елемент е различен от нула.
- При краткото логическо "ИЛИ" a || b, ако a е "True", тогава целият израз е "True" и b не се пресмята.

Употребяват се също вградени функции, които съответстват на действието на логическите оператори:

- **and**(*rel1, rel2*) – еквивалентно на *rel1 & rel2*;
- **or**(*rel1, rel2*) – еквивалентно на *rel1 | rel2*;
- **no**t(*rel*) – еквивалентно на *~rel*.

MATLAB разполага с много вградени логически функции, които се използват за записване на логически изрази. Такива са например функциите **any, all, exist, isempty, isequal, isfinit, isinf, isnan, isreal, isscalar, isvector** и др. Едни от най-използваните са **any** и **all**:

- **any**(*v*) – истина, ако сред елементите на вектора v има поне един различен от нула;
- **all**(*v*) – истина, ако всички елементи са различни от нула;
- **any**(*logexpr*(v)) – истина, ако поне един елемент на вектора v удовлетворява логическото условие *logexpr*(v);
- **all**(*logexpr*(v)) – истина, ако всички елементи на v удовлетворяват условието *logexpr*(v).

Ако аргумент на горните функции е матрица, тя се обработва по стълбове, като резултатът е вектор-ред от единици и нули. За да се извърши обработката по редове трябва да се добави втори аргумент, равен на 2, например **any**(A, 2). В този случай резултатът е вектор-стълб.

Функциите, чиито имена започват с английския глагол **is** (е ли?) проверяват дали е изпълнено съответното условие или дали аргументът им е от съответния тип.

Примери:

```
>> v = [5, 7, 0, 8, 2];
>> any(v)
ans =
     1
```

```
>> all(v)
ans =
     0

>> any(v > 10)
ans =
     0

>> all(v > - 1)
ans =
     1
```

```
>> A = [1 2 3 4

0 1 2 3
0 0 1 2
0 0 0 0];
```

```
>> any(A)
ans =
      1 1 1 1
```

```
>> any(A, 2)

ans =
     1
     1
     1
     0

>> all(A)
ans =
     0 0 0 0

>> all(A, 2)
ans =
     1
     0
     0
     0
```

```
>> v1 = [1 2 3]; v2 = v1;
>> isequal(v1, v2) % Еквивалентни ли са v1 и v2 ?
ans =
     1
```

3.5. Превключващ оператор switch

Този оператор се използва, когато се налага да се изпълняват различни клонове от програмата в зависимост от стойността на даден израз, т.е. да се реализира многовариантно разклонение. Общият вид на този оператор е следния:

```
switch expr
    case val1                % В случай че expr == val1, изпълни
    statements1;
    case val2                % В случай че expr == val2, изпълни
    statements2;
    case {val3, val4}
    statements34;
    ................
    otherwise
    statements;
end
```

В този запис **switch, case, otherwise** и **end** са системни ключови думи. Действието на оператора е следното: ако изразът *expr* има стойност *val1* се изпълняват оператори *statements1*, след което управлението се предава на оператора след **end**; ако изразът *expr* има стойност *val2*, тогава се изпълняват оператори *statements2* и т. н. Възможно е няколко стойности на израза *expr* да се обединяват с помощта на главни скоби " { } ". В този случай, ако *expr* има една от двете стойности *val3* или *val4* (логическо ИЛИ), системата ще изпълни операторите *statements34*. Ако изразът *expr* не приеме нито една от указаните стойности, тогава се изпълняват операторите *statements*, разположени между ключовите думи **otherwise** и **end.** Няма никакви ограничения за броя на ключовите думи **case**. Ключовата дума **otherwise** (аналог на **else** в една **if**-конструкция) не е задължителна.

Изразът *expr* трябва да приема само цели числени стойности или да е име на низ. Очевидно това условие се удовлетворява и от логически функции, които могат да имат само стойности 1 или 0. В тази конструкция не могат да се използват изрази, чиито стойности са числа с плаваща точка, защото между такива числа е невъзможно да се прилага релацията логическо равенство " == ".

Действието на оператор **switch** на практика е аналогично на действието на оператор **if** в третия му вариант.

Пример: Да се състави програма, с която въведената с цифра оценка да се извежда с текст.

Програма:

```
ocenka=input('въведи оценка ');
switch ocenka
    case 2, disp('слаб');
    case 3, disp('среден');
    case 4, disp('добър') ;
    case 5, disp('много добър');
    case 6, disp('отличен');
    otherwise
    disp('не е въведена коректна оценка');
end
```

Програмата е съхранена като файл **ocenkap.m** и се изпълнява, като в командния ред се задава името на файла.

```
>> ocenkap
въведи оценка 5
много добър
```

3.6. Аритметичен оператор за цикъл for

Операторът **for** се използва за организиране на цикъл в програмата. Броят на изпълненията на тялото на цикъла е предварително известен и определен. Общият вид на оператора е следния:

```
for index = inival : step : endval
    statements;
end
```

където:
index - управляващата променлива на цикъла;
inival – началната стойност на променливата *index;*
step – стъпката, с която се изменя променливата *index*;
endval – крайната стойност на променливата *index*.

За разлика от повечето езици за програмиране, в MATLAB не е задължително управляващата променлива *index* да приема целочислени стойности. Освен това е възможно началната стойност на управляващата променлива да е по-голяма от крайната, т. е. *inival > endval*, като в този случай стъпката *step* трябва да е отрицателна.

Действието на оператора е следното: за всички стойности на *index* от началната *inival* до крайната *endval* със стъпка на промяна *step* се изпълняват операторите *statements*. След това се продължава с изпълнение на оператора след **end** .

Пример:

```
>> for i=8:-2:4
disp(i)
end

     8

     6

     4
```

Параметърът *step* не е задължителен. Ако този параметър липсва, по подразбиране се приема *step* = 1.

В MATLAB е възможен и следният вариант на оператора за цикъл **for**:

```
for i = V
    statements;
end
```

В този случай *V* може да бъде вектор или матрица. Ако *V* е вектор, променливата *i* приема последователно за стойности елементите на *V*. Ако *V* е матрица, тогава *i* е вектор-стълб, който последователно приема стойностите на отделните стълбове на матрицата.

Примери:

```
>> v=[1 2 3];
>> for i=v
disp(i)
end

     1

     2

     3
```

```
>> mat=[1 2 3; 1 2 3];
>> for i=mat
disp(i)
end

     1
     1

     2
     2

     3
     3
```

Операторът **for** се използва много често, когато трябва да се организира начертаване на графики на семейство криви, което се описва от дадена функция за различни стойности на нейни параметри. Такъв пример е разгледан в Тема 5.

Пример: Да се изчисли сумата на първите *n* естествени числа.

Задачата се свежда до изброяването на целите числа от 1 до *n* и прибавянето им към текущо изчислената сума (т.е. извършване *n* на брой пъти на действие сумиране). Същият пример е разгледан в темата за циклични алгоритми, където е представена и блок-схема на алгоритъма.

Въвеждат се следните помощни величини (променливи). На тях съответстват клетки в оперативната памет на компютъра.

s - клетка от оперативната памет, в която ще се натрупва сумата на числата от 1 до *n*;

i - управляваща променлива на цикъла. Променливата *i* последователно приема всички целочислени стойности от 1 до *n*, с което "управлява" повторението на тялото на цикъла.

Програма:

```
n=input('въведете брой на числата n=');
s=0;
for i=1:n
    s=s+i;
end
disp(['сумата на числата е s=', num2str(s)])
```

Програмата е съхранена като файл **sumnp.m** и се изпълнява, като в командния ред се задава името на файла.

```
>> sumnp
въведете брой на числата n=7
сумата на числата е s=28
```

Пример: Да се намери сумата на положителните елементите на матрица R(M, N).

Вариант 1: Броят на редовете M и стълбовете N се въвежда от потребителя

```
M=input('въведете броя на редовете на матрицата ');
N=input('въведете броя на стълбовете на матрицата ');
R=input('въведете елементите на матрицата ');
s=0;
for i=1:M
    for j=1:N
        if R(i,j)>0
            s=s+R(i,j);
        end
    end
end
disp(['сума на положителните елементи ', num2str(s)])
```

Програмата е съхранена като файл **sumpolp.m** и се изпълнява, като в командния ред се задава името на файла.

```
>> sumpolp
въведете броя на редовете на матрицата 2
въведете броя на стълбовете на матрицата  3
въведете елементите на матрицата [1 -2 -3 ;-3 -4 6]
сума на положителните елементи 7
```

Вариант 2: Броят на редовете M и стълбовете N се определя в програмата с функцията **size**

```
R=input('въведете елементите на матрицата ');
[M, N]= size(R)
s=0;
for i=1:M
    for j=1:N
        if R(i,j)>0
            s=s+R(i,j);
        end
    end
end
disp(['сума на положителните елементи ', num2str(s)])
```

Програмата е съхранена като файл **matrix1.m** и се изпълнява, като в командния ред се задава името на файла.

```
>>matrix1
въведете елементите на матрицата [1 2 3; -1 -1 -1]
```

```
M =

     2


N =

     3

сума на положителните елементи 6
```

3.7. Условен оператор за цикъл **while**

Операторът за цикъл **while** се използва за организиране на цикли, които се изпълняват докато е в сила дадено условие. В този случай броят на изпълнението на тялото на цикъла не е предварително известен. Общият вид на оператора **while** е следният:

```
while logexpr
    statements;
end
```

Възможно е записване на оператора на един ред:

```
while logexpr, statements, end
```

Действието на оператора е следното: операторите *statements* се изпълняват циклично дотогава, докато логическият израз *logexpr* има стойност True. Когато логическият израз *logexpr* получи стойност False управлението се предава на оператора след end .

Пример: Да се изчисли сумата на първите *N* естествени числа като се използва цикъл с предусловие. Примерът е разгледан в темата за циклични алгоритми, където е показана и блок-схема на алгоритъма.

Програма:

```
N=input('въведи брой на числата N=');
S=0;
I=1;
while I<=N
    S=S+I;
    I=I+1;
end
disp(['сумата на числата е S=', num2str(S)]);
```

Програмата е съхранена като файл **sum2p.m** и се изпълнява, като в командния ред се задава името на файла.

```
>> sum2p
въведи брой на числата N=7
сумата на числата е S=28
```

Пример: Да се построи таблица на функциите sin(x), sqrt(x) и exp(x) за стойности на аргумента x в интервала $0.1 \le x \le \pi$ със стъпка $\pi / 5$.

Програма:

```
x = 0.1; % начална стойност на аргумента x
dx = pi/5; % стъпка, с която нараства x
disp(['      x       sin(x)    sqrt(x)  exp(x)'])
while x <= pi
    disp([x, sin(x), sqrt(x), exp(x)]) % извеждане в 4 колони
    x = x + dx; % поредно нарастване на аргумента x
end
```

Програмата е съхранена като файл **tabulacia.m** и се изпълнява, като в командния ред се задава името на файла.

```
>>tabulacia
      x       sin(x)    sqrt(x)  exp(x)
    0.1000    0.0998    0.3162    1.1052

    0.7283    0.6656    0.8534    2.0716

    1.3566    0.9772    1.1647    3.8831

    1.9850    0.9155    1.4089    7.2787

    2.6133    0.5041    1.6166   13.6436
```

3.8. Оператори break, continue, pause, keyboard и return

Тези оператори се използват за управление на изпълнението на една програма.

Операторите **break** и **continue** най-често се използват съвместно с оператора **if** в тялото на цикъл **for** или **while.** Операторът **break** прекратява изпълнението на цикъла и предава управлението на операторите, разположени след оператора **end**. При това, ако цикълът е вграден в друг цикъл, продължава изпълнението на външния цикъл.

Операторът **continue** предизвиква пропускане на операторите, разположени между него и оператора **end.** С това изпълнението на цикъла продължава.

Операторът **pause** прекратява временно изпълнението на програмата. Ако е въведен без аргумент, паузата продължава, докато бъде натиснат произволен клавиш от клавиатурата. Обикновено се използва за разглеждане на резултатите от пресмятанията.

При въвеждане на оператора с аргумент – **pause**(*n*), паузата трае *n* секунди, след което системата сама продължава изпълнението на програмата. Числото *n* може да съдържа до две цифри след десетичната точка, т. е. можем да задаваме времето за пауза до стотни от секундата. Когато се използва оператор **pause** без аргумент, е полезно пред него да разположите следния оператор, който извежда подсещащо съобщение на екрана:

```
>>disp(' Pause. Press any key to continue !')
```

Операторът **keyboard** предава временно управлението на клавиатурата, при което се появява промптът K>>. В този режим могат да се извършат произволни действия – проверка и промяна на данни, стартиране на функции и процедури на MATLAB и др. Връщането към програмата става, като подадем командата K>> **return**. Изпълнението на програмата продължава с оператора, намиращ се непосредствено след оператора **keyboard**.

Операторът **return** се използва и за преждевременно завръщане в извикващата програма.

3.9. Организиране на менюта

При създаване на програми, които предоставят на потребителя избор на един от няколко варианта, е удобно този избор да се направи в диалогов режим. Добър подход в такива случаи е възможните варианти да се представят като меню.

Един от възможните варианти за представяне на меню е да се използват операторите **disp** и **switch**. За съставяне на меню може да се използва следния шаблон:

```
disp(' Заглавие на менюто ')
disp('****************************')
disp(' Вариант 1')
disp(' Вариант 2')
disp(' Вариант 3')
.........
n = input(' Въведете номера на желания вариант: ');
switch n
    case 1  % Вариант 1
    statements1;
    case 2  % Вариант 2
    statements2;
    case 3  % Вариант 3
    statements3;
    ..................
end
```

Пример: По зададен радиус да се изчисли: дължината на окръжността (вариант 1), или лицето на кръга (вариант 2), или обема на сфера (вариант 3).

```
disp(' изчисление с радиус')
```

```
disp(' въведете 1. - дължина на окръжност')
disp(' въведете 2. - лице на кръг')
disp(' въведете 3. - обем на сфера')
n = input(' Въведете номера на желания вариант: ');
r = input(' Въведете радиус ');
switch n
    case 1  % Вариант 1
    c=2*pi*r;
    disp(['дължината на окръжност с радиус r=', num2str(r) , '
e c=', num2str(c) ])

    case 2  % Вариант 2
    s=pi*r*r;
    disp(['лицето на кръг с радиус r=', num2str(r) , ' e s=',
num2str(s) ])

    case 3  % Вариант 3
    v=(4*pi*r^3)/3;
    disp(['обемът на сфера с радиус r=', num2str(r) , ' e v=',
num2str(v) ])
end
```

В MATLAB е включена специална функция за организиране на меню.

```
n = menu('Заглавие','Вариант 1','Вариант 2', ...);
```

Предназначението на тази функция е да организира менюто и избора на вариант от него. За реализиране на предложените варианти в програмата отново се използва оператор **switch**.

При изпълнението на оператор, който съдържа функцията меню, на екрана се извежда специален панел за избор. Желаният вариант се избира с натискане на съответния бутон.

Пример: Реализиране на предишния пример с функция menu.

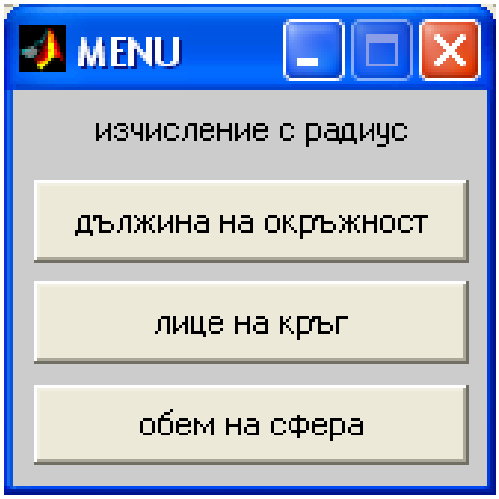
```
n = menu(' изчисление с радиус ', 'дължина на окръжност',
'лице на кръг', 'обем на сфера');
r = input(' Въведете радиус ');
switch n
    case 1  % Вариант 1
    c=2*pi*r;
    disp(['дължината на окръжност с радиус r=', num2str(r) , '
    e c=', num2str(c) ])

    case 2  % Вариант 2
    s=pi*r*r;
    disp(['лицето на кръг с радиус r=', num2str(r) , ' e s=',
    num2str(s) ])

    case 3  % Вариант 3
    v=(4*pi*r^3)/3;
```

```
    disp(['обемът на сфера с радиус r=', num2str(r) , ' е v=',
    num2str(v) ])
end
```

При изпълнение на програмата, на екрана се появява панел, който е представен на Фиг.3.2. От този панел се избира вариант, след което в командния прозорец се извежда съответният резултат.



Фиг.3.2. Панел за избор на вариант