3. Программирование в **Octave**

3.1 Основные операторы языка программирования ***Octave.***

Рассмотренные в предыдущих главах группы команд, состоящие из операторов присваивания и обращения к встроенным функциям, представляют собой *простейшие программы* Octave. Если такая программа хранится в файле с расширением .m (.M), то для ее выполнения достаточно в командной строке Octave ввести имя этого файла (без расширения). В Octave встроен достаточно мощный язык программирования. Рассмотрим основные операторы этого языка и примеры их использования.

3.1.1 Оператор присваивания

Оператор присваивания служит для определения новой переменной (п. 2.2). Для того, чтобы определить новую переменную, достаточно присвоить ей значение:

```
имя переменной = значение выражения
```

Любую переменную Octave воспринимает как матрицу. В простейшем случае матрица может состоять из одной строки и одного столбца (листинг 3.1).

```
>>> m=pi
m =  3.1416
>>> m
m =  3.1416
>>> m(1)
ans =  3.1416
>>> m(1,1)
ans =  3.1416
>>> m(1,2)
error: A(I): Index exceeds matrix dimension.
>>> m(3)
error: A(I): Index exceeds matrix dimension.
>>> M=e;M(3,3)=e/2;
>>> M
M =
   2.71828   0.00000   0.00000
   0.00000   0.00000   0.00000
   0.00000   0.00000   1.35914
```

Листинг 3.1

3.1.2 Организация простейшего ввода и вывода в диалоговом режиме

Даже при разработке простейших программ возникает необходимость ввода исходных данных и вывода результатов. Если для вывода результатов на экран можно просто не ставить «;» после оператора, то для *ввода исходных данных* при разработке программ, работающих в диалоговом режиме, следует использовать функцию

```
имя переменной = input('подсказка');
```

Если в тексте программы встречается оператор `input`, то выполнение программы приостанавливается, Octave выводит на экран текст `подсказки` и переходит в режим ожидания ввода. Пользователь вводит с клавиатуры значение и нажимает клавишу Enter. Введенное пользователем значение будет присвоено переменной, имя которой указано слева

от знака присваивания.

Для *вывода результатов* можно использовать функцию следующей структуры:

`disp('строка символов')` или `disp(имя переменной)`

ПРИМЕР 3.1. Создать программу для вычисления значения *y* по формуле *y*=*sin*(*x*), при заданном значении *x*.

Текст программы и результаты ее работы показаны в листинге 3.2.

```
x=input('Введите значение x=');
y=sin(x);
disp('Значение y=');disp(y);
>>>%Результат работы программы
>>>Введите значение x= pi/4
Значение y=
0.70711
```

Листинг 3.2

3.1.3 Условный оператор

Одним из основных операторов, реализующим ветвление в большинстве языков программирования, является *условный оператор*. Существует обычная, сокращенная и расширенная формы этого оператора языке программирования Octave 7.

Обычный условный оператор имеет вид:

```
if условие
      операторы1
else
      операторы2
end
```

Здесь `условие` – логическое выражение, `операторы1`, `операторы2` – операторы языка или встроенные функции Octave. Обычный оператор `if` работает по следующем алгоритму: если `условие` истинно, то выполняются `операторы1`, если ложно – `операторы2`.

ПРИМЕР 3.2. Даны вещественные числа *x* и *y*. Определить принадлежит ли точка с координатами (*x*; *y*) заштрихованной части плоскости (рис. 3.1).

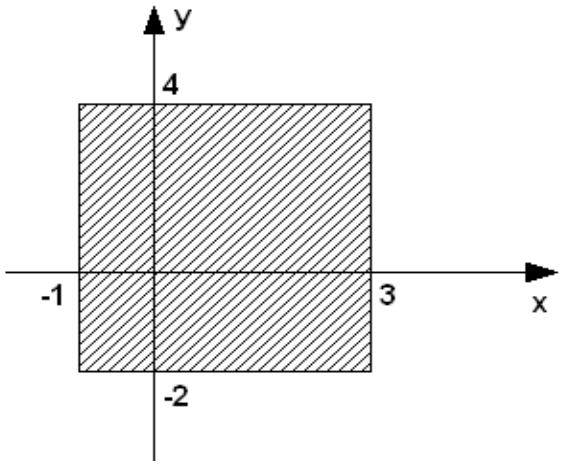

Рис. *3.1:* Графическое представление задачи из примера Ошибка: источник перёкрестной ссылки не найден

Как показано на рис. 3.1 плоскость ограничена линиями *x*=-1, *x*=3, *y*=-2 и *y*=4. Значит

точка с координатами (x; y) будет принадлежать этой плоскости, если будут выполняться следующие условия: $x \geq -1$, $x \leq 3$, $y \geq -2$ и $y \leq 4$. Иначе точка лежит за пределами плоскости.

Далее приведен текст программы и результаты ее работы.

```
x=input('x=');
y=input('y=');
if (x>=-1) & (x<=3) & (y>=-2) & (y<=4)
 disp('Точка принадлежит плоскости')
else
 disp('Точка не принадлежит плоскости');
end
>>>%Результаты работы программы
>>>x= 3
y= 3
Точка принадлежит плоскости
>>>%----------------------------
>>>x= 4
y= 4
Точка не принадлежит плоскости
```

Листинг 3.3

Сокращенный условный оператор записывают так:

```
if условие
      операторы
end
```

Работает этот оператор следующим образом. Если `условие` истинно, то выполняются `операторы`, в противном случае управление передается оператору следующему за оператором `if` (листинг 3.4).

```
z=0;
x=input('x=');
y=input('y=');
if (x~=y)
 z=x+y;
end;
disp('Значение Z=');
disp(z);
>>>%Результаты работы программы
>>>x= 3
y= 5
Значение Z= 8
>>>x= 3
y= 3
Значение Z= 0
```

Листинг 3.4

Расширенный условный оператор применяют когда одного условия для принятия решения недостаточно:

```
if условие1
      операторы1
elseif условие2
      операторы2
elseif условие 3
      операторы3
…
```

```
elseif условие n
      операторыn

else
      операторы
end
```

Расширеный оператор `if` работает так. Если `условие1` истинно, то выполняются `операторы1`, иначе проверяется `условие2`, если оно истинно, то выполняются `операторы2`, иначе проверяется `условие3` и т.д. Если ни одно из условий по веткам `elseif` не выполняется, то выполняются операторы по ветке `else`.

Рассмотрим использование расширенного условного оператора на примере.

ПРИМЕР 3.3. Дано вещественное число *x*. Для функции, график которой приведен на рис. 3.2 вычислить *y=f*(*x*).

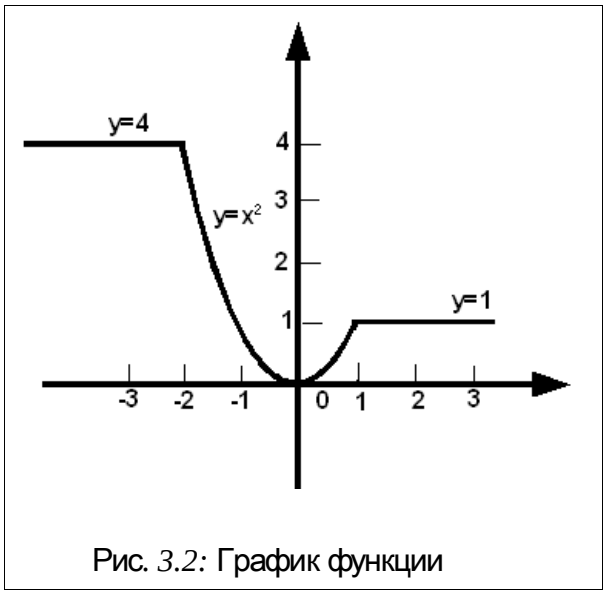

Рис. *3.2:* График функции

Аналитически функцию представленную на рис. 3.2 можно записать так:

$$y(x)=\begin{Bmatrix} 4, & x\leqslant -2 \\ 1, & x\geqslant 1 \\ x^2, & -2<x<1 \end{Bmatrix}$$

Составим словесный алгоритм решения этой задачи:

1. Начало алгоритма.
2. Ввод числа `x` (аргумент функции).
3. Если значение `x` меньше либо равно `-2`, то переход к п. 4, иначе переход к п. 5.
4. Вычисление значения функции: `y=4`, переход к п. 8.
5. Если значение `x` больше либо равно `1`, то переход к п. 6, иначе переход к п. 7.
6. Вычисление значения функции: `y=1`, переход к п. 8.
7. Вычисление значения функции: `y=x`^{2}.
8. Вывод значений аргумента `x` и функции `y`.
9. Конец алгоритма.

Текст программы будет иметь вид:

```
x=input('x=');
if x<=-2
      y=4;
elseif x>=1
      y=1;
```

```
else
      y=x^2;
end;
disp('y=');disp(y);
>>>%Результаты работы программы
>>>x= 2
y= 1
>>>%------------------------------
>>>x= -3
y= 4
>>>%------------------------------
>>>x= 0.5
y=  0.25000
```

Листинг 3.5

3.1.4 Оператор альтернативного выбора

Еще одним способом организации разветвлений является оператор альтернативного выбора следующей структуры:

```
swith параметр
 case значение1
      операторы1
 case значение2
      операторы2
 case значение3
      операторы3
…
 otherwise
      операторы
end
```

Оператор `switch` работает следующим образом: если значение `параметра` равно `значениию1`, то выполняются `операторы1`, иначе если `параметр` равен `значениию2`, то выполняются `операторы2`; в противном случае, если значение `параметра` совпадает со `значением3`, то выполняются `операторы3` и т.д. Если значение `параметра` не совпадает ни с одним из значений в группах `case`, то выполняются `операторы`, которые идут после служебного слова `otherwise`.

Конечно, любой алгоритм можно запрограммировать без использования `switch`, используя только `if`, но использование оператора альтернативного выбора `switch` делает программу более компактной.

Рассмотрим использование оператора switch на примере решения следующем примере.

ПРИМЕР 3.4. Вывести на печать название дня недели, соответствующее заданному числу D, при условии, что в месяце 31 день и 1-е число – понедельник.

Для решения задачи воспользуемся функцией mod, позволяющей вычислить остаток от деления двух чисел, и условием, что 1-е число – понедельник. Если в результате остаток от деления заданного числа D на семь будет равен единице, то это понедельник, двойке – вторник, тройке – среда и так далее. Следовательно, при построении алгоритма необходимо использовать семь условных операторов. Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта (листинг 3.6).

```
D=input('Введите число от 1 до 31   ');
%Вычисление остатка от деления D на 7,
%сравнение его с числами от 0 до 6.
```

```
switch mod(D,7)
case 1
      disp('ПОНЕДЕЛЬНИК')
case 2
      disp('ВТОРНИК')
case 3
      disp('СРЕДА')
case 4
      disp('ЧЕТВЕРГ')
case 5
      disp('ПЯТНИЦА')
case 6
      disp('СУББОТА')
otherwise
      disp('ВОСКРЕСЕНЬЕ')
end
```

Листинг 3.6

3.1.5 Условный циклический оператор

Оператор *цикла с предусловием* в языке программирования Octave имеет вид:

```
while выражение
      операторы
end
```

Работает цикл с предусловием следующим образом. Вычисляется значение `выражения`. Если оно истинно, выполняются `операторы`. В противном случае цикл заканчивается, и управление передается оператору, следующему за телом цикла. `Выражение` вычисляется перед каждой итерацией цикла. Если при первой проверке `выражение` ложно, цикл не выполнится ни разу. `Выражение` должно быть переменной или логическим выражением.

ПРИМЕР 3.5. Дано натуральное число *N*. Определить количество цифр в числе.

Для того, чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело. Например, пусть `N=12345`, тогда количество цифр `kol = 5`. Результаты вычислений сведены в таблицу 3.1.

Табл. *3.1.* Определение количества цифр числа

kol	N
1	12345
2	12345 div10=1234
3	1234 div 10=123
4	123 div 10=12
5	12 div 10=1
	1 div 10=0

Текст программы, реализующей данную задачу, можно записать так:

```
N=input('N=');
M=N;        %Сохранить значение переменной N.
kol=1;      %Пусть число состоит из одной цифры.
while round(M/10) > 0
%Выполнять тело цикла, пока число делится нацело на 10.
      kol=kol+1;          %Счетчик количества цифр
      M=round(M/10);      %Изменение числа.
```

```
end
disp('kol=');disp(kol);
%Результат работы программы
>>>N= 12345678
kol=
8
```

Листинг 3.7

3.1.6 Оператор цикла с известным числом повторений

Для записи цикла с известным числом повторений применяют оператор

```
for параметр = начальное значение:шаг:конечное значение
      операторы
end
```

Выполнение цикла начинается с присвоения `параметру` цикла `начального значения`. Затем следует проверка, не превосходит ли `параметр` цикла `конечное значение`. Если результат проверки утвердительный, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняются `операторы` в цикле. Далее параметр меняет свое значение на значение `шага` и снова производится проверка значения `параметра` цикла, и алгоритм повторяется.

Если шаг цикла равен 1, то оператор записывают так

```
for параметр = начальное значение:конечное значение
      операторы
end
```

ПРИМЕР 3.6. Дано натуральное число *N*. Определить *K* – количество делителей этого числа, не превышающих его (Например, для *N*=12 делители 1, 2, 3, 4, 6. Количество *K*=5).

Для решения поставленной задачи нужно реализовать следующий алгоритм: в переменную `K`, предназначенную для подсчета количества делителей заданного числа, поместить значение, которое не влияло бы на результат, т.е. нуль. Далее организовать цикл, в котором изменяющийся параметр `i` выполняет роль возможных делителей числа `N`. Если заданное число делится нацело на параметр цикла, это означает, что `i` является делителем `N`, и значение переменной `K` следует увеличить на единицу. Цикл необходимо повторить `N/2` раз.

```
N=input('N=');
K=0;
for i=1:round(N / 2)
%Если N делится нацело на i, то
if mod(N, i) == 0
K=K+1;     %увеличить счетчик на единицу.
end
end
disp(' K=');disp(K);
%Результат работы программы
>>>N= 12
 K= 5
```

Листинг 3.8

3.1.7 Операторы передачи управления

Операторы передачи управления принудительно изменяют порядок выполнения команд. В языке программирования Octave таких операторов два. Операторы `break` и

`continue` используют только внутри циклов. Так оператор `break` осуществляет немедленный выход из циклов `while`, `for` и управление передается оператору, находящемуся непосредственно за циклом. Оператор `continue` начинает новую итерацию цикла, даже если предыдущая не была завершена.

ПРИМЕР 3.7. Дано натуральное число *N*. Определить, является ли оно простым. Натуральное число *N* называется простым, если оно делится нацело без остатка только на единицу и *N*. Число 13 – простое, так как делится только на 1 и 13, N=12 не является простым, так как делится на 1, 2, 3, 4, 6 и 12.

Алгоритм решения этой задачи заключается в том, что число N делится на параметр цикла `i`, изменяющийся в диапазоне от `2` до `N/2`. Если среди значений параметра не найдется ни одного числа, делящего заданное число нацело, то `N` – простое число, иначе оно таковым не является. Разумно предусмотреть в программе два выхода из цикла. Первый – естественный, при исчерпании всех значений параметра, а второй — досрочный, с помощью оператора `break`. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра.

Текст программы приведен в листинге 3.9.

```
N=input('Введите число ');
% Предполагаем, что число N является простым (pr=1).
pr=1;
% Перебираем все возможные делители числа N от 2 до N/2.
for i=2:N/2
% Если N делится на i,
      if mod(N,i)==0
% то число N не является простым (pr=0)
           pr=0;
% и прерывается выполнение цикла.
           break;
      end
end
% Если pr равно 1, то N - простое число.
if pr==1
      disp('ПРОСТОЕ ЧИСЛО')
% Если pr равно 0, то N - не является простым.
else
      disp('НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ')
end
>>>% Результаты работыпрограммы
>>>Введите число  12
НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ
>>>%---------------------------
>>>Введите число  13
ПРОСТОЕ ЧИСЛО
```

Листинг 3.9

3.2 Обработка массивов и матриц

Octave содержит достаточное количество операций предназначенных для работы с векторами и матрицами (см. гл. 6). В этом параграфе мы остановимся на поэлементной обработке одномерных и двумерных массивов.

Ввод массивов и *матриц* следует организовывать поэлементно, например так:

```
N=input('N='); %Ввод элементов массива
```

```
for i=1:N
x(i)=input(strcat('x(',int2str(i),')='));
end
>>>% Результат работы программы
>>>N= 5
x(1)= 1
x(2)= 2
x(3)= 3
x(4)= 4
x(5)= 5
%Ввод элементов матрицы
N=input('N=');
M=input('M=');
for i=1:N
for j=1:M
a(i,j)=input(strcat('a(',int2str(i),',',int2str(j),')='));
end
end
>>>% Результат работы программы
>>>N= 3
M= 3
a(1,1)= 1
a(1,2)= 2
a(1,3)= 3
a(2,1)= 4
a(2,2)= 5
a(2,3)= 6
a(3,1)= 7
a(3,2)= 8
a(3,3)= 9
```

Листинг 3.10

Для вывода приглашений вида `x(i)=` и `a(i,j)=` в функции `input` использовались функции работы со строками: `strcat(s1, s2, …, sn)` и `int2str(d)` . Функция `strcat` предназначена для объединения строк `s1`, `s2`, …, `sn` в одну строку, которая и возвращается в качестве результата. Функция `num2str` преобразовывает число `d` в строку символов.

Алгоритм *вычисления суммы элементов массива* достаточно прост. В переменную предназначенную для накапливания суммы записывают ноль (`s=0`), затем добавляют к `s` первый элемент массива и результат записывают в переменную `s`, далее к переменной `s` добавляют второй элемент массива и результат записывают в `s` и далее аналогично добавляем к `s` остальные элементы массива.

```
s=0;
for i=1:N
      s=s+x(i);
end
```

Листинг 3.11

При нахождении *суммы элементов матрицы* последовательно суммируют элементы всех строк.

```
s=0;
for i=1:N
```

```
 for j=1:M
      s=s+a(i,j);
 end
end
```

Листинг 3.12

Алгоритм вычисления *произведения элементов массива* следующий: на первом шаге начальное значение произведения равно 1 (`p=1`), затем последовательно умножают `p` на очередной элемент, и результат записывают в `p`.

```
p=1;
for i=1:N
      p=p*x(i);
end
```

Листинг 3.13

При вычислении *произведения элементов матрицы* последовательно перемножают элементы всех строк.

```
p=1;
for i=1:N
 for j=1:M
      p=p*a(i,j);
 end
end
```

Листинг 3.14

Алгоритм решения задачи *поиска максимума* и его номера в массиве следующий. Пусть в переменной с именем `Max` хранится значение максимального элемента массива, а в переменной с именем `Nmax` – его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную `Max`, а в `Nmax` – его номер (то есть 1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную `Max`, а в переменную `Nmax`– текущее значение индекса `i`.

На листинге 3.15 представлен фрагмент программы поиска максимума.

```
Max=a(1);
Nmax=1;
for i=1:N
 if x(i)>Max
      Max=x(i);
      Nmax=i;
 end;
end;
```

Листинг 3.15

Алгоритм *поиска минимального элемента* в массиве будет отличаться от приведенного выше лишь тем, что в конструкции `if` текста программы знак поменяется с > на <.

В листинге 3.16 приведен фрагмент программы, реализующий алгоритм *поиска минимального элемента матрицы* и *его индексов*.

```
Min=a(1,1);
Nmin=1;
Lmin=1;
for i=1:N
 for j=1:M
      if a(i,j)<Min
           Min=a(i,j);
```

```
            Nmin=i;
            Lmin=j;
      end;
 end;
end;
```

Листинг 3.16

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Рассмотрим наиболее известный алгоритм сортировки *методом пузырька*. Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Те же действия выполним для второго и третьего, третьего и четвертого, `i`–го и (`i+1`)–го, (`n−1`)–го и `n`–го элементов. В результате этих действий самый большой элемент станет на последнее (`n`-е) место. Теперь повторим данный алгоритм сначала, но последний (`n`-й) элемент рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на (`n`–1)-е место. Так повторяем до тех пор, пока не упорядочим по возрастанию весь массив. Фрагмент программы *сортировки элементов массива по убыванию* приведен ниже.

```
for i=1:N-1
 for j=1:N-i
      if x(j)>x(j+1)
            b=x(j);
            x(j)=x(j+1);
            x(j+1)=b;
      end;
 end;
end;
```

Листинг 3.17

Для *сортировки по убыванию* нужно в операторе `if` заменить знак > на <.

Рассмотрим *удаление элемента из массива*. Пусть необходимо удалить из массива `x`, состоящего из `n` элементов, `m`–й по номеру элемент. Для этого достаточно записать элемент (`m+1`) на место элемента `m`, (`m+2`) – на место (`m+1`) и т.д., `n` – на место (`n−1`) и при дальнейшей работе с этим массивом использовать `n−1` элемент. В листинге 3.18 приведен фрагмент программы, реализующий этот алгоритм.

```
for i=m:1:n-1
      x(i)=x(i+1);
end;
```

Листинг 3.18

В Octave есть встроенные функции вычисления суммы (`sum`), произведения (`prod`) элементов массива (матрицы), поиска максимума (`max`) и минимума (`min`), сортировки (`sort`), однако только понимание алгоритмов работы функций позволит решать нестандартные задачи обработки массивов и матриц. Рассмотрим решение нескольких практических задач.

ПРИМЕР 3.8. Найти наименьшее простое число в массиве $x(n)$, если таких чисел несколько, определите их количество.

Листинге содержит программу решения этой задачи с подробными комментариями.

```
% Ввод размера массива.
N=input('N=');
% Цикл для ввода элементов массива.
for i=1:N
      x(i)=input(strcat('x(',int2str(i),')='));
```

```
end
% Переменная pr=0 (в массиве не обнаружено простых чисел).
% Если pr=0 -простых чисел нет, pr=1, простые числа есть.
pr=0;
for i=1:N
% Переменная L используется при проверке,
%является ли данный элемент массива x(i) простым числом,
%L=1, пока не встретились делители числа,
% L станет равным 0, если встретятся делители числа.
      L=1;
% Цикл по j от 2 до x(i)/2 для проверки является ли
%число простым (поиск возможных делителей числа).
      for j=2:x(i)/2
% Если x(i) делится на j, то встретился делитель числа,
%x(i) не является простым, в L=0 и выходим из цикла по j
%с помощью оператора break.
           if mod(x(i),j)==0
                L=0;
                break;
           end;
      end;
% Проверяем значение переменной L, если L=1,
%то число x(i) - простое.
      if L==1
% Если число x(i) - простое и при этом pr=0,
%то это означает, что встретилось первое простое число,
           if pr==0
% и его записываем в переменную min,
% т.е. предполагаем, что x(i) и является минимальным простым,
                min=x(i);
% количество минимумов равно 1,
                k=1;
% записываем в pr 1, т.к. в массиве есть простые числа.
                pr=1;
% Иначе, если pr=1, т.е. встретилось очередное (не первое)
% простое число,
           else
% сравниваем x(i) с min, если x(i)<min,
                if x(i)<min
% этот элемент x(i) записываем в переменную min,
                     min=x(i);
% количество минимумов равно 1.
                     k=1;
                else
% Если очередной элемент x(i) равен min,
                     if x(i)==min
% то количество минимумов увеличивается.
                          k=k+1;
                     end;
                end;
           end;
```

```
      end;
end;
% Если после перебора всех элементов массива,
%переменная pr осталась равной 0 (простых чисел нет),
if pr==0
% то вывод соответствующего сообщения.
      disp('Простых чисел нет!!!!')
% Если были простые числа,
else
% то вывод min (минимальное простое число)
%и k (количество минимумов).
      disp(min);
      disp(k);
end;
```

Листинг 3.19

ПРИМЕР 3.9. В квадратной матрице A(N,N) обнулить столбцы, в которых элемент на побочной диагонали является максимальным.

Алгоритм решения этой задачи состоит в следующем: в каждом столбце находим максимальный элемент и проверяем, если наибольший элемент расположен на побочной диагонали, то обнуляем все элементы в том столбце. Элемент находится на побочной диагонали, если его номер строки `i` и номер столбца `j` связаны соотношением `i+j=n+1`.

Листинге 3.20 содержит текст программы для решения поставленной задачи.

```
% Ввод размера квадратной матрицы.
N=input('N=');
% Ввод квадратной матрицы.
for i=1:N
 for j=1:N
 A(i,j)=input(strcat('a(',int2str(i),',',int2str(j),')='));
 end
end
% Цикл по всем столбцам матрицы, в каждом из которых ищем
%максимальный элемент и его номер.
for j=1:N
% Предполагаем, что первый элемент в столбце
% является максимальным,
      max=A(1,j);
% В переменную nmax, в которой будет
% храниться номер максимального
% элемента j-го столбца записываем 1.
      nmax=1;
% В цикле по i перебираем все элементы j-го столбца.
      for i=2:N
% Если очередной элемент больше max,
           if A(i,j)>max
% то в переменную max записываем этот элемент,
                max=A(i,j);
% а в переменную nmax – номер строки, где находится элемент.
                nmax=i;
           end;
      end;
% Если в текущем столбце максимальный элемент
```

```
%находится на побочной диагонали,
 if nmax==N+1-j
% то обнуляем все элементы в этом столбце.
      for i=1:N
           A(i,j)=0;
      end;
 end;
end;
```

Листинг 3.20

3.3 Обработка строк в *Octave*

В языке программирования Octave есть множество функций работы со строками. Рассмотрим некоторые из них.

Табл. *3.2.* Функции для работы со строками.

Функция	**Описание функции**	**Пример использования**
`char(code)`	Возвращает символ по его коду `code`	`>>> char(100)` `ans = d` `>>> char(80:85)` `ans = PQRSTU`
`deblank(s)`	Формируется новая строка путем удаления пробелов в конце строки `s`	`>>> deblank('OCTAVE  ')` `ans = OCTAVE`
`int2str(x)`	Преобразование чисел, хранящихся в массиве (матрице) `x` к целому типу и запись результатов в массив символов	`>>> int2str(123.456)` `ans = 123` `>>> int2str([9.8 6.9])` `ans = 10 7`
`findstr(str,substr)`	Возвращает номер позиции, начиная с которой подстрока `substr` входит в строку `str`	`>>> Str='Visual C++';` `>>> S='C++';` `>> findstr(Str,S)` `ans =8`
`lower(s)`	Возвращает строку путем преобразования строки `s` к строчным буквам	`>>> S='QtOctave';` `>>> lower(S)` `ans = qtoctave`
`mat2str(x,n)`	Преобразовывает числовую матрицу `x` в строку; если присутствует необязательный параметр `n`, то перед преобразованием в строку все элементы матрицы округляются до `n` значащих цифр в числе	`>>> X=[7.895; -9.325]` `X =` `7.8950` `-9.3250` `>>> mat2str(X)` `ans =` `[7.8949999999999996;` `-9.3249999999999993]` `>>> mat2str(X,2)` `ans = [7.9;-9.3]`
`num2str(x,n)`	Преобразовывает числовую матрицу (массив) `x` в массив символов с точностью 4 цифры после	*`>>>X=[7.89578;`* *`-9.32985]`* `>>> num2str(X)` `ans =`

Функция	Описание функции	Пример использования
	десятичной точки, если присутствует необязательный параметр n, то перед преобразованием в строку все элементы матрицы округляются до n цифр в числе	7.8958 -9.3299 >>> num2str(X,2) ans = 7.9 -9.3 >>> num2str(X,1) ans = 8 -9
sprintf(format, x)	Формирует строку из чисел, хранящихся в числовой переменной x в соответствии с форматом format.	*>>> x=789.65432145;* >>> sprintf('X=%4.2e',x) ans = X=7.90e+02 >>> y=-654.12345678; >>> sprintf('Y=%7.3f',y) ans = Y=-654.123
sscanf(s, format)	Функция возвращает из строки s числовое значение или массив значений в соответствии с форматом	>>> s='1234.5' s = 1234.5 >>> x=sscanf(s, '%f') x = 1234.5 >>> x=sscanf(s, '%d') x = 1234
str2double(s)	Формирование числа из строки s, если это возможно	*>>> s='1.456e-2';* >>> str2double(s) ans = 0.014560
str2num(s)	Формирование массива чисел из строки (массива символов) s	*>>> s='-pi 2 1.6';* >>> str2num(s) ans = -3.1416 2.0000 1.6000
strcat(s1,s2,…sn)	Формируется строка путем объединения строк s1, s2, …, sn	>>> s1='Octave'; >>> s2='Qt'; >>> s=strcat(s2,s1) s = QtOctave ;
strcmp(s1,s2)	Возвращает 1, если строки s1 и s2 совпадают, 0 – в противном случае	>>> S1='Пример'; >>> strcmp(S1,'Пример') ans = 1 >>> S2='1-е Мая'; >>> S3='1-е мая'; >>> strcmp(S2,S3) ans = 0
strcmp(s1,s2)	Сравнение строк s1 и s2, не различая строчные и прописные буквы	>>> S2='1-е Мая'; >>> S3='1-е мая'; >>> strcmp(S2,S3) ans = 1
strjust(s,direction)	Выравнивание строки s в соответствии с направлением direct: right – выравнивание по	>>>S='Pascal 7.0 '; >>>strjust(S,'right') ans = Pascal 7.0

Функция	Описание функции	Пример использования
	правому краю, `left` – выравнивание по левому краю, `center` – выравнивание по центру	`>> S='  Pascal 7.0';` `>>strjust(S,'left')` `ans = Pascal 7.0` `>> S='  Pascal 7.0';` `>>strjust(S,'center')` `ans = Pascal 7.0`
`strncmp(s1,s2,n)`	Сравнение первых `n` символов строк `s1` и `s2`, возвращает 1, если первые `n` символов строк `s1` и `s2` совпадают, 0 – в противном случае	`>>> S1='Мама мыла раму';` `>>> S2='Мама мыла Петю';` `>>> strncmp(S1,S2,10)` `ans = 1` `>>> strncmp(S1,S2,11)` `ans = 0`
`strrep(s,subs,subsnew)`	Формирует новую строку из строки `s` путем замены подстрок `subs` на подстроки `subsnew`	`>>>S='07. 07. 2007';` `>>> strrep(S,'7','8')` `Ans =` `08. 08. 2008`
`strtok(s,delimiter)`	Поиск первой подстроки в строке `s`, отделённой пробелом или символом табуляции (при отсутствии параметра `delimiter`) или первой подстроки, отделенной от `s` одним из символов, входящих в `delimiter`. Функция может возвращать 2 параметра: первый – найденная подстрока, второй – содержит остаток строки `s` после `strtok`	*`>>>S='Винни-Пух`* *`и Пятачок';`* `>>> strtok(S)` `ans = Винни-Пух` *`>>> S='Привет, Пух!';`* `>>> strtok(S)` `ans = Привет,` `>>>[S1,S2]=strtok(S,',')` `S1 = Привет` `S2 = , Пух!`
`upper(s)`	Возвращает строку `s` преобразованную к прописным буквам	*`>>> S='Octave';`* `>>> upper(S)` `ans = OCTAVE`

3.4 Работа с файлами в *Octave*

Octave предоставляет широкие возможности для работы с текстовыми и двоичными файлами. *Текстовыми* называют файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «`конец строки`». Конец самого файла обозначается символом «`конец файла`». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.

В *двоичных файлах* информация считывается и записывается в виде блоков определенного размера, в них могут храниться данные любого вида и структуры.

Операции с файлами в Octave имеют много общего с функциями обработки файлов в языке Си. Читатель, имеющий опыт программирования на Си, сможет убедиться, что фрагменты Си-программ обработки файлов могут с минимальными изменениями быть перенесены в Octave.

3.4.1 Обработка текстовых файлов

Для начала работы с текстовым файлом его необходимо *открыть*, для чего в Octave используется функция следующей структуры

```
fopen(filename, mode)
```

Здесь `filename` – строка, в которой хранится полное имя открываемого файла, `mode` – строка, которая определяет режим работы с файлом. Параметр `mode` может принимать следующие значения:

'`rt`' – открываемый текстовый файл используется в режиме чтения;

'`rt+`' – открываемый текстовый файл используется в режиме чтения и записи;

'`wt`' – создаваемый пустой текстовый файл предназначен только для записи информации;

'`wt+`' – создаваемый пустой текстовый файл предназначен для чтения и записи информации;

'`at`' – открываемый текстовый файл будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;

'`at+` – открываемый текстовый файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан.

Функция `fopen` возвращает идентификатор файла (номер, присвоенный файлу). Существует три стандартных системных файла: стандартный ввод (`stdin`), стандартный вывод (`stdout`) и файл, отвечающий за вывод сообщений об ошибках (`stderr`), за которыми закреплены идентификаторы 0, 1 и 2 соответственно.

Для форматированного вывода информации в файл можно использовать функцию следующего вида:

```
fprintf(f, s1, s2).
```

Здесь `f` – идентификатор файла (значение идентификатора возвращается функцией `fopen`), `s1` – строка вывода, `s2` – список выводимых переменных.

В строке вывода вместо выводимых переменных указывается строка преобразования следующего вида:

```
%[флаг][ширина][.точность] тип.
```

Значения основных параметров строки преобразования (символы управления форматированием) приведены в табл. 3.3.

Табл. *3.4*. Символы управления форматированием

Параметр	Назначение
	Флаги
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «−»
0	Заполнение незаполненные позиции дополняются нулями
	Ширина
n	Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции дополняются пробелами
	Точность
ничего	Точность по умолчанию
m	Для типов *e*, *E*, *f* выводить m знаков после десятичной точки
	Тип
c	При вводе символьный тип char, при выводе один байт.

Параметр	Назначение
d	Десятичное целое со знаком
i	Десятичное целое со знаком
o	Восьмеричное целое без знака
u	Десятичное целое без знака
x, X	Шестнадцатеричное целое без знака, при x используются символы a - f, при X – A-F.
f	Значение со знаком вида [-]dddd.dddd
e	Значение со знаком вида [-]d.dddd e[+|-]ddd
E	Значение со знаком вида [-]d.dddd E[+|-]ddd
g	Значение со знаком типа e или f в зависимости от значения и точности
G	Значение со знаком типа E или F в зависимости от значения и точности
s	Строка символов

В строке вывода могут использоваться некоторые специальные символы, приведенные в табл. 3.5.

Табл. *3.5.* Специальные символы

Символ	Назначение
\b	Сдвиг текущей позиции влево
\n	Перевод строки
\r	Перевод в начало строки, не переходя на новую строку
\t	Горизонтальная табуляция
\'	Символ одинарной кавычки
\''	Символ двойной кавычки
\?	Символ ?

При *считывании данных из файла* можно воспользоваться функцией следующего вида:

```
A=fscanf(f, s1, n).
```

Здесь `f` – идентификатор файла, который возвращается функцией `fopen`, `s1` – строка форматов вида `%[ширина][.точность]тип`, `s2` – имя переменной, количество считываемых значений.

Функция `fscanf` работает следующим образом: из файла с идентификатором `f` считывается в переменную `A` `n` значений в соответствии с форматом `s1`. При чтении числовых значений из текстового файла следует помнить, что два числа считаются разделенными, если между ними есть хотя бы один пробел, символ табуляции или символ перехода на новую строку.

При считывании данных из текстового файла пользователь может следить, достигнут ли конец файла с помощью функции `feof(f)` (`f` – идентификатор файла), которая возвращает единицу, если достигнут конец файла, и ноль в противном случае.

После выполнения всех операций с файлом он должен быть закрыт с помощью функции:

```
fclose(f).
```

Здесь `f` – идентификатор закрываемого файла. С помощью функции `fclose('all')` можно закрыть сразу все открытые файлы, кроме стандартных системных файлов.

Рассмотрим использование рассмотренных выше функций на простых примерах.

ПРИМЕР 3.10. Поменять местами элементы, расположенные на главной и побочной диагонали квадратной матрицы *A*(*N*,*N*). Исходную и преобразованную матрицы вывести в текстовый файл *prim_3_10.txt*.

Далее приведена программа решениями задачи с комментариями.

```
N=input('N=');% Ввод размеров матрицы.
```

```
% Ввод элементов матрицы.
for i=1:N
 for j=1:N
 A(i,j)=input(strcat('A(',int2str(i),',',int2str(j),')='));
 end
end
% Открыть файл  для записи (создать новый пустой файл).
f=fopen('prim_4_9.txt','wt');
% Вывод в файл строки ИСХОДНАЯ МАТРИЦА A
% и перевод курсора на новую  строку (символ \n).
fprintf(f,'ИСХОДНАЯ МАТРИЦА A\n');
% Цикл для построчной записи элементов матрицы в файл.
for i=1:N
% Цикл для поэлементной записи в файл i-й строки матрицы.
 for j=1:N
%Запись очередного элемента A(i,j) и символа табуляции в файл.
      fprintf(f,'%f\t',A(i,j));
 end
% После записи очередной строки переход
%к следующей строке файла.
 fprintf(f,'\n');
end
% В каждой строке матрице
for i=1:N
% поменять местами элементы,
%расположенные на главной и побочной диагоналях.
      b=A(i,i);
      A(i,i)=A(i,N+1-i);
      A(i,N+1-i)=b;
end
% Вывод в файл строки МАТРИЦА A после преобразования
% и перевод курсора на новую строку (символ \n).
fprintf(f,'МАТРИЦА A после преобразования\n');
% Двойной цикл для вывода матрицы в файл.
for i=1:N
 for j=1:N
      fprintf(f,'%f\t',A(i,j));
 end
 fprintf(f,'\n');
end
% Закрытие файла после записи в него необходимой информации.
fclose(f);
```

Листинг 3.21

В результате работы программы создан файл *prim_3_10.txt* (рис. 3.3), который можно открыть при помощи обычного текстового редактора.

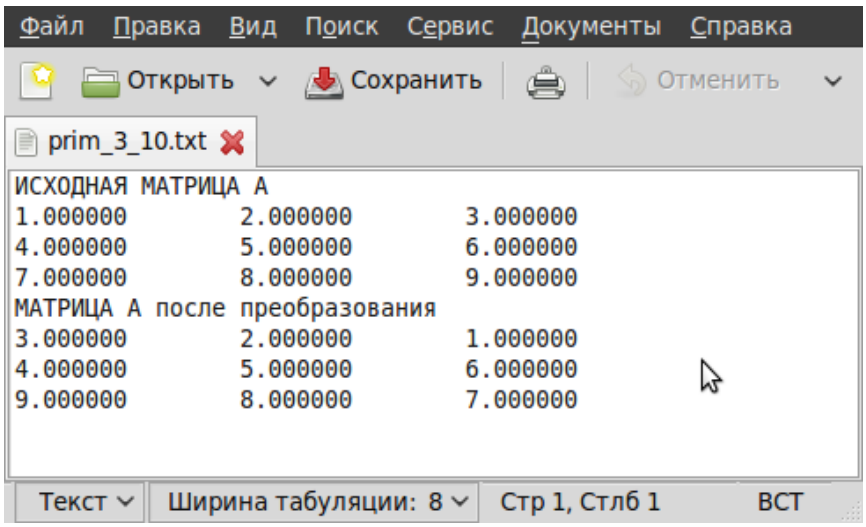

Рис. *3.3:* Результат работы программы из примера *3.10*

Обратите внимание, что после записи информации в файл, его обязательно надо закрывать с помощью функции `fclose`. Дело в том, что `fprintf` не обращается непосредственно к диску – он пишет информацию в специальный участок памяти, называемый буфером файла. После того как буфер заполнится, вся информация из него вносится в файл. При вызове функции `fclose` сначала происходит запись буфера файла на диск, и только потом файл закрывается. Если файл не закрыть, то он автоматически закрывается при завершении работы программы, но при этом пропадает информация, хранимая в буфере файла.

ПРИМЕР 3.11. Записать матрицу A(N, M) в файл следующим образом. Пусть в первой строке текстового файла хранятся числа N и M, а затем – построчно матрица A. Листинг 3.22 содержит фрагмент программы для создания подобного файла.

```
% Ввод размеров матрицы.
N=input('N=');
M=input('M=');
% Ввод элементов матрицы.
for i=1:N
 for j=1:M
 A(i,j)=input(strcat('A(',int2str(i),',',int2str(j),')='));
 end
end
% Открыть файл  для записи.
f=fopen('primer.txt','wt');
% Записать в файл N и M, разделив их символом табуляции,
% после чего перейти на новую строку в файле.
fprintf(f,'%d\t%d\n',N,M);
% Цикл для построчной записи элементов матрицы в файл.
for i=1:N
% Цикл для поэлементной записи в файл i-й строки матрицы.
 for j=1:M
%Запись очередного элемента A(i,j) и символа табуляции в файл.
      fprintf(f,'%g\t',A(i,j));
 end;
% После записи очередной строки переход к следующей строке.
```

```
 fprintf(f,'\n');
end;
% Закрыть файл.
fclose(f);
```

Листинг 3.22

После выполнения этой программы будет создан текстовый файл *prim_3_11.txt* (рис. 3.4).

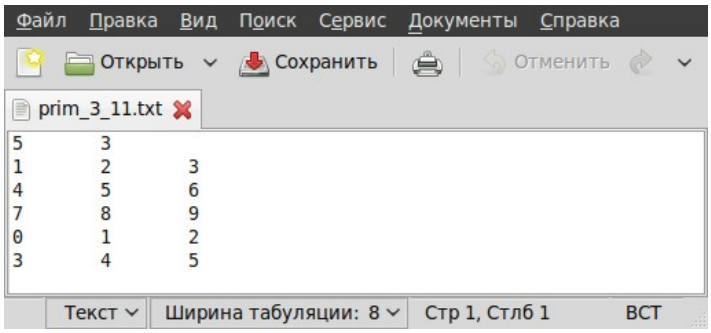

Рис. *3.4:* Результат работы программы из примера *3.11*

ПРИМЕР 3.12. Считать информацию из файла *prim_3_11.txt* в матрицу. Рассмотрим поэлементное (листинг 3.23) и построчное (листинг 3.24) чтение матрицы из файла. В обоих случаях чтение из текстового файла начинается с чтения значений `N` и `M`, которые хранятся в первой строке файла *prim_3_11.txt*. Затем при построчном чтении организован цикл, в котором считывается одна строка с помощью функции `fscanf`. При поэлементном чтении организован двойной цикл, в котором функция `fscanf` считывает значение одного элемента матрицы из файла.

```
f=fopen('primer.txt','rt');% Открываем файл для чтения.
% Считываем одно целое число
% в переменную N (количество строк матрицы).
N=fscanf(f,'%d',1);
% Считываем одно целое число
% в переменную M (количество столбцов матрицы).
M=fscanf(f,'%d',1);
% Двойной цикл по строкам и столбцам.
for i=1:N
 for j=1:M
% Считываем один элемент из файла в матрицу.
      A(i,j)=fscanf(f,'%g',1);
 end;
end;
fclose(f);% Закрываем файл.
A % Вывод матрицы на экран
% Результат работы программы
>>>A =
1 2 3
4 5 6
7 8 9
0 1 2
3 4 5
```

Листинг 3.23

```
% ------------------------------------------------
% Открываем файл для чтения.
f=fopen('primer.txt','rt');
% Считываем одно целое число
% в переменную N (количество строк матрицы).
N=fscanf(f,'%d',1);
% Считываем одно целое число
% в переменную M (количество столбцов матрицы).
M=fscanf(f,'%d',1);
% Открываем цикл по строкам.
for i=1:N
% Считываем в i-ю строку M элементов матрицы (всю строку).
      A(i,:)=fscanf(f,'%g',M);
end;
% Закрываем файл.
fclose(f);
A % Вывод матрицы на экран
% Результат работы программы
>>>A =
1 2 3
4 5 6
7 8 9
0 1 2
3 4 5
```

Листинг 3.24

ПРИМЕР 3.13. В текстовом файле *one.txt* хранятся вещественные значения (рис. 3.5). Считать их в массив.

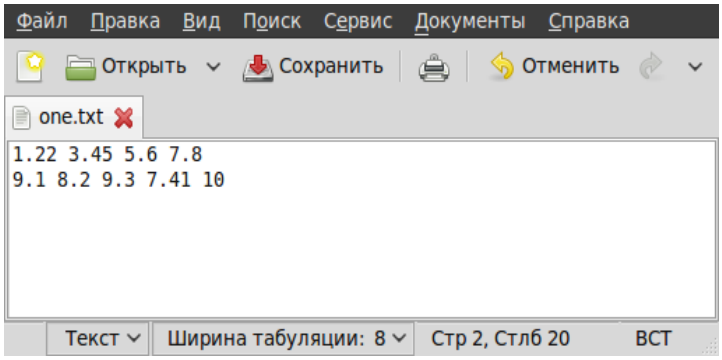

Рис. *3.5:* Текстовый файл *one.txt*

Возможно считывание из файла всего массива целиком (листинг 3.25) или поэлементное считывание данных из файла (листинг 3.26).

```
f=fopen('one.txt');% Открываем файл для чтения.
% Считываем содержимое файла целиком в массив.
x=fscanf(f,'%f');
% Сформирован массив x.
% Результат работы программы
>>>x =
1.2200
3.4500
5.6000
```

```
7.8000
9.1000
8.2000
9.3000
7.4100
10.0000
```

Листинг 3.25

```
% ------------------------------------------------------
f=fopen('one.txt');% Открываем файл для чтения.
% В переменной i будет храниться номер элемента массива,
% куда будет осуществляться считывание очередного значения
% из файла, в начале в i записываем 0,
% пока в массиве нет элементов.
i=0;
while ~ feof(f) % Проверяем, если не достигнут конец файла,
i=i+1;          % то увеличиваем i,
% и считываем очередной i-й элемент из файла в массив
X(i)=fscanf(f,'%f',1);
end
% В массиве x из i элементов будут храниться
% все числа из файла one.txt
X
% Результат работы программы
>>>X = 1.220 3.450 5.600 7.800 9.100 8.200 9.300 7.410  10.000
```

Листинг 3.26

Обратите внимание, если данные в файле располагаются в несколько строк, то программы, аналогичные приведенным на листингах 3.23, 3.24, считывают их как матрицу значений, а программы, приведенные на листингах 3.25, 3.26 считывают значения из файла в одномерный массив.

В языке программирования Octave есть функции для записи и чтения матриц в текстовый файл и из текстового файла.

Функция `dlmread` предназначена для *чтения числовых данных из текстового файла* в матрицу. Существуют четыре варианта использования функции.

1. `M=dlmread('filename')` – чтение чисел из текстового файла `filename` в матрицу `M`, числа внутри строки отделяются запятой. В листинге 3.27 представлен результат чтения данных из файла *one.txt* (рис. 3.5).

```
>>> H=dlmread('one.txt')
H =
    1.22000    3.45000    5.60000    7.80000    0.00000
    9.10000    8.20000    9.30000    7.41000   10.00000
```

Листинг 3.27

Если в каких-либо строках текстового файла пропущено значение, то недостающий элемент матрицы будет равен нулю.

2. `M=dlmread('filename', delimiter)` – чтение чисел из текстового файла `filename` в матрицу `M`, числа отделяются символом, хранящемся в `delimiter`. Например, `M=dlmread('ab.txt', '\t')` – чтение из файла *ab.txt* чисел в матрицу `M`, внутри строки числа отделяются табуляцией. В качестве примера рассмотрим файл *two.txt* (рис. 3.6), в котором числа внутри строки разделены двоеточием. В листинге 3.28 представлен результат чтения данных из файла *two.txt*.

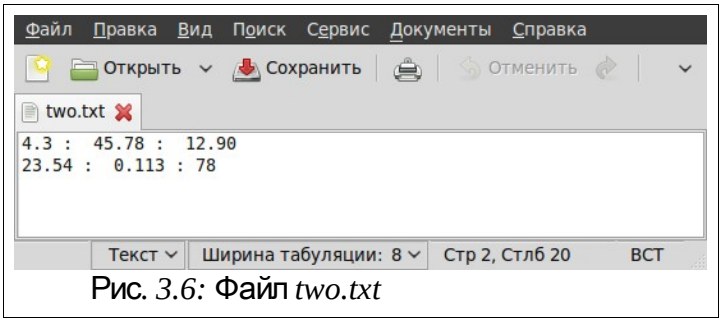

Рис. *3.6:* Файл *two.txt*

```
>>> L=dlmread('two.txt',':')
L =
    4.30000   45.78000   12.90000
   23.54000    0.11300   78.00000
```

Листинг 3.28

3. `M=dlmread('filename', delimiter, R, C)` – чтение чисел из текстового файла `filename` в матрицу `M`, числа внутри строки отделяются символом, хранящимся в `delimiter`, начиная со строки `R` и столбца `C`. Строки и столбцы нумеруются с 0. Листинг 3.29 представлено чтение матрицы из файла *two.txt* (рис. 3.6) начиная со второй строки и третьего столбца.

```
>>> L=dlmread('two.txt',':',1,2)
L =  78
```

Листинг 3.29

4. `M = dlmread ('filename', delimiter, range)` – чтение чисел из текстового файла `filename` в матрицу `M`, числа внутри строки отделяются символом, хранящимся в `delimiter`, `range` определяет область [`row_start col_start row_end col_end`], `row_start`, `col_start` - левый верхний угол, а `row_end`, `col_end` - правый нижний угол области данных, которые считываются из файла. Область `range` может быть задана в стиле электронных таблиц, например – '`A1:C3`'. В листинге 3.30 представлено чтение матрицы из файла *one.txt* (рис. 3.5).

```
>>> P=dlmread('one.txt',' ',[0 0 1 2])
P =
1.2200 3.4500 5.6000
9.1000 8.2000 9.3000
>>> P=dlmread('one.txt',' ','A1:C2')
P =
1.2200 3.4500 5.6000
9.1000 8.2000 9.3000
```

Листинг 3.30.

Функция `dlmwrite` предназначена для записи матрицы в текстовый файл. Существуют три варианта использования функции.

1. `dlmwriter('filename',M)` – запись матрицы `M` в текстовый файл `filename`, числа внутри строки отделяются запятой.

2. `dlmwriter('filename', M, delimiter)` – запись матрицы `M` в текстовый файл, числа внутри строки отделяются символом, хранящимся в `delimiter`.

3. `dlmwriter('filename', M, delimiter, R, C)` – запись матрицы `M`, начиная со строки `R` и столбца `C` в текстовый файл `filename`, числа внутри строки отделяются символом, хранящимся в `dlmwriter`.

Вывести содержимое текстового файла с именем `filename` на экран можно с помощью функции:

```
type('filename');
```

Листинг 3.31 содержит пример вызова функций `dlmwrite` и `type`.

```
>>> M=[1 2 3;4 5 6; 7 8 9];
>>> dlmwrite('file.txt',M);
>>> type('file.txt')
file.txt is the user-defined function defined from: ./file.txt
1,2,3
4,5,6
7,8,9
```

Листинг 3.31

3.4.2 Обработка двоичных файлов в **Octave**

Двоичный файл, как и текстовый открывается с помощью функции `fopen`. Разница в том, что в параметре `mode` вместо буквы `t` должна использоваться буква `b` (от слова `binary` – двоичный): «`rb+`», «`wb+`» и т.д.

Чтение из двоичного файла осуществляется с помощью обращения к функции

```
[A n]= fread (f, n, type);
```

где `f` – идентификатор файла, `n` – количество считываемых из файла элементов, `type` – тип считываемых из файла элементов.

Возможные значения параметра `type` приведены в табл. 3.6.

Табл. *3.6.* Возможные значения параметра *type*

Параметр type	**Размер элемента в байтах**	**Описание**
`uchar`	1	Целое число без знака (0÷255)
`schar`	1	Целое со знаком (-128÷127)
`int16`	2	Целое со знаком (-32768÷32767)
`int32`	4	Целое со знаком (-2147483648÷2147483647)
`int64`	8	Целое со знаком $-(2^{63}-1)\div(2^{63}-1)$
`uint16`	2	Целое без знака (0÷65535)
`uint32`	4	Целое без знака (0÷4294967295)
`uint64`	8	Целое без знака $(0\div 2^{64}-1)$
`float32`	4	Вещественное число (3.4E-38÷3.4E+38)
`float64`	8	Вещественное число (1.7E-308÷1.7E+308)

Функция `fread` считывает из предварительно открытого файла с идентификатором `f` `n` элементов типа `type` и записывает их в массив (матрицу) `A`, количество реально считанных элементов возвращается в переменной `n`. Если при обращении к функции `fread` отсутствует параметр `type`, то подразумевается что из двоичного файла будут считываться значения типа `uchar` (однобайтовое целое без знака). Если пропущен и параметр `n`, то в массив `A` будут считываться все значения до конца файла. Параметр `n` может быть представлен в виде `[m k]`, в этом случае данные считываются в матрицу размером `m` на `k`.

Файл – последовательная структура данных. После открытия файла доступен первый элемент, хранящийся в файле. После чтения очередной порции данных указатель файла смещается на следующую порцию данных.

Текущая позиция указателя файла (смещения от начала файла в байтах) возвращается функцией

```
ftell(f);
```

Здесь `f` – идентификатор уже открытого с помощью `fopen` файла.

Для *перемещения указателя в начало файла* служит функция

```
frewind(f);
```

Функция

```
fseek(f, n, origin);
```

обеспечивает все остальные *перемещения указателя файла*. Функция перемещает текущую позицию в файле с идентификатором `f` на `n` байт, относительно позиции `origin`.

Параметр `origin` может принимать одно из следующих значений:

строка '`bof`' или число -1 определяет смещение относительно начала файла, в этом случае значение n может быть только положительным;

строка '`eof`' или число 1 определяет смещение относительно конца файла на `n` байтов назад, в этом случае значение n также должно быть положительным;

строка '`cof`' или число 0 определяет смещение относительно текущей позиции на `n` байтов вперед (`n`>0) или назад (`n`<0).

Запись в двоичный файл осуществляется с помощью функции

```
n=fwrite(f, A, type),
```

где `f` – идентификатор файла, `A` – массив (матрица) значений, `type` – тип записываемых в файл элементов. Функция `fwrite` записывает в заранее открытый с идентификатором `f` массив `A`, и возвращает количество реально записанных в файл значений.

Рассмотрим несколько примеров работы с двоичными файлами.

ПРИМЕР 3.14. Создать двоичный файл *abc.dat*, куда записать целое число `N`, а затем `N` вещественных чисел. Решить эту задачу можно двумя способами:

1. Записать в файл целое число `N`, а затем в цикле `N` вещественных чисел.

2. Записать в файл целое число N, а затем одним оператором `fwrite` записать в файл массив из `N` вещественных чисел.

Решение задачи с комментариями первым способом представлено на листинге 3.32, вторым – на листинге 3.33.

```
% Ввод значения переменной N.
N=input('N=');
% Открытие нового двоичного файла abc.dat в режиме записи.
f=fopen('abc.dat','wb');
% Запись числа N в двоичный файл abc.dat.
fwrite(f,N,'int16');
% Цикл для ввода N вещественных чисел и записи их
% в двоичный файл abc.dat
for i=1:N
% Ввод очередного вещественного числа x.
x=input('X=');
% Запись очередного числа x в двоичный файл abc.dat.
fwrite(f,x,'float32');
end;
% Закрытие файла.
fclose(f);
```

Листинг 3.32

```
% ---------------------------------------
% Ввод значения переменной N.
N=input('N=');
% Открытие нового двоичного файла abc.dat в режиме записи.
f=fopen('abc.dat','wb');
% Запись числа N в двоичный файл abc.dat.
fwrite(f,N,'int16');
% Цикл для ввода массива из N вещественных чисел.
```

```
for i=1:N
% Ввод очередного вещественного числа в массив x.
x(i)=input(strcat('x(',int2str(i),')='));
end;
% Запись массива вещественых x в двоичный файл abc.dat.
fwrite(f,x,'float32');
% Закрытие файла.
fclose(f);
```

Листинг 3.33

В результате будет сформирован двоичный файл размером N*4+2 байт. Запустим любую из этих программ на выполнение в командной строке, введем N=20 и сформируем файл abc.dat размером 82 байта.

ПРИМЕР 3.15. Считать данные из файла *abc.dat*, сформированного в задаче из примера 3.14 в массив вещественных чисел.

Программа решения этой задачи представлена на листинге 3.34.

```
% Открытие двоичного файла abc.dat в режиме чтения.
f=fopen('abc.dat','rb');
% Чтение числа N из двоичного файла abc.dat.
N=fread(f,1,'int16');
% Чтение массива из N вещественных
% чисел из двоичного файла abc.dat.
x=fread(f,N,'float32');
% Закрытие файла.
fclose(f);
```

Листинг 3.34

ПРИМЕР 3.16. Считать данные из файла *abc.dat*, сформированного в задаче из примера 3.14 в матрицу вещественных чисел. Зная, что в файле *abc.dat* хранится 20 чисел, в качестве примера запишем их матрицу размером 4x5. Программа решения этой задачи представлена в листинге 3.35. В результате работы этой программы будет сформирована матрица вещественных чисел G(4,5).

```
% Открытие двоичного файла abc.dat в режиме чтения.
f=fopen('abc.dat','rb');
% Чтение числа N из двоичного файла abc.dat.
N=fread(f,1,'int16');
% Чтение матрицы вещественных чисел G(4,5)
% из двоичного файла abc.dat.
G=fread(f,[4 5],'float32');
% Закрытие файла.
fclose(f);
>>> G
G =
1.20000 9.00000 7.40000 8.90000 5.40000
3.40000 0.10000 6.50000 0.90000 4.30000
5.60000 9.20000 5.60000 8.70000 3.20000
7.80000 8.30000 7.80000 6.50000 2.10000
```

Листинг 3.35

Отдельную задачу представляет чтение данных из двоичного файла, если заранее не известно количество элементов в файле. В листинге 3.36 представлена программа, с помощью которой можно создать файл вещественных чисел.

```
% Ввод значения переменной N.
N=input('N=');
```

```
% Открытие нового двоичного файла abc2.dat в режиме записи.
f=fopen('abc2.dat','wb');
% Цикл для ввода N вещественных чисел и записи
% их в двоичный файл abc2.dat
for i=1:N
% Ввод очередного вещественного числа x.
x=input('X=');
% Запись очередного числа x в двоичный файл abc.dat.
fwrite(f,x,'float32');
end;
fclose(f);
```

Листинг 3.36

Отличие этой программы от представленных на листингах 3.34, 3.35 состоит в том, что количество записанных в файл вещественных чисел *abc2.dat* в нем не хранится. Поэтому чтение данных из такого файла осуществляется несколько по-другому. Известно, что функция `ftell(f)` возвращает текущее положение указателя файла. Если с помощью функции `fseek(f,0,1)` передвинуть указатель в конец файла, а затем обратиться к функции `ftell(f)`, можно вычислить количество байт в файле. Разделив полученное число на 4 (размера вещественного числа типа '`float32`'), получим количество элементов в файле, после чего считаем нужное количество элементов в массив с помощью функции `fread`. Программа, реализующая описанные выше действия, представлена на листинге 3.37.

```
% Открытие двоичного файла abc2.dat в режиме чтения.
f=fopen('abc2.dat','rb');
% Перевод указателя в конец файла.
fseek(f,0,1);
% С помощью функции ftell вычисляем количество байт в файле,
% после чего делим на размер одного элемента,
% для 'float32' это число 4.
N=ftell(f)/4;
% Переводим указатель на начало файла
frewind(f);
% Чтение из двоичного файла abc2.dat в массива x
% N вещественных чисел.
x=fread(f,N,'float32');
% Закрытие файла.
fclose(f);
```

Листинг 3.37

Аналогичным образом можно будет считать данные любого типа из двоичного файла. Отличие будет состоять только в том, что в операторе `N=ftell(f)/4;` необходимо заменить число 4 на действительный размер элементов, хранящихся в файле и при обращении к функции `fread` указать в качестве третьего параметра реальный тип считываемых данных. Функция `fread` не считает больше элементов, чем находится в файле, независимо от того, что указано во втором параметре. Поэтому, если в листинге 3.37 оператор вычисления `N` записать следующим образом `N=ftell(f)`, то программа будет корректно считывать данные в массив любого типа. Будет происходить следующее: оператор `x=fread(f,N,type)` будет пытаться считать `N` элементов из двоичного файла, но не считает значений больше, чем их там есть, остановится на конце файла.

3.5 Функции в *Octave*

В Octave файлы с расширением .m могут содержать не только тексты программ (группа операторов и функций Octave), но и могут быть оформлены как отдельные функции. В этом случае имя функции должно совпадать с именем файла, в котором она хранится (например, функция с именем `primer` должна храниться в файле *primer.m*).

Функция в Octave имеет следующую структуру.

Первая строка функции это *заголовок*:

```
function [y1,y2,…yn]=name_function(x1,x2,…,xm)
```

Здесь `name_function` – имя функции, `x1`, `x2`, …, `xm` – список входных параметров функции, `y1`, `y2`, …, `yn` – список выходных параметров функции. Функция заканчивается служебным словом `end`. Таким образом, в простейшем случае структуру функции можно записать следующим образом:

```
function [y1,y2,…yn]=name_function(x1,x2,…,xm)
      оператор1;
      оператор2;
      …
      операторk;
end
```

В файле с расширением *.m*, кроме основной функции, имя которой совпадает с именем файла, могут находиться так называемые *подфункции*. Эти функции доступны только внутри файла. Таким образом, общую структуру функции можно представить так:

```
%Здесь начинается основная функция m-файла,
% имя которой должно совпадать с именем файла,
% в котором она хранится
function [y1,y2,…yn]=name_function(x1,x2,…,xm)
% Среди операторов основной функции могут быть операторы
% вызова подфункций f1, f2, f3, …,fl
оператор1;
оператор2;
…
операторk;
end;   %  здесь заканчивается основная функция
function [y1,y2,…yn]=f1(x1,x2,…,xm) % начало первой подфукции
      операторы
end % конец первой подфункции
function [y1,y2,…yn]=f2(x1,x2,…,xm) % начало второй подфукции
      операторы
end % конец второй подфункции
…
function [y1,y2,…yn]=fn(x1,x2,…,xm) % начало n-й подфукции
      операторы
end % конец n-й подфункции
```

Такая структура, близка к структуре программ на языке Си. Она не допускает вложенности функций друг в друга. Однако в Octave возможен и другой синтаксис, в котором разрешено использование вложенных функций, поэтому структура основной функции может быть и такой:

```
function [y1,y2,…yn]=name_function(x1,x2,…,xm)
 function [y1,y2,…yn]=f1(x1,x2,…,xm) % начало первой подфукции
      операторы
 end % конец первой подфункции
```

```
 function [y1,y2,…yn]=f2(x1,x2,…,xm) % начало второй подфукции
      операторы
 end % конец второй подфункции
…
 function [y1,y2,…yn]=fn(x1,x2,…,xm) % начало n-й подфукции
      операторы
 end % конец n-й подфункции
 оператор1;
 оператор2;
 …
 операторk;
end   %  здесь заканчивается основная функция
```

Рассмотрим пример.

ПРИМЕР 3.17. Написать функцию, предназначенную для удаления из массива $x(N)$ простых чисел.

Функцию назовем `udal_prostoe`. Ее входными данными являются: числовой массив `x`; `N` – количество элементов в массиве. Выходными данными функции `udal_prostoe` будут: массив `x`, из которого удалены простые числа; новый размер массива `N` после удаления из него простых чисел.

В функции `udal_prostoe` будут использоваться две вспомогательные функции: функция `prostoe`, которая проверяет, является ли число простым; функция `udal` удаления элемента из массива.

Заголовок основной функции имеет вид: `function [x N]=udal_prostoe(x, N)`

Заголовок подфункции запишем так: `function pr=prostoe(P)`

Функция `prostoe` проверяет, является ли число `P` простым, она возвращает 1, если `P` – простое, 0 – в противном случае.

Заголовок подфункции `udal` имеет вид: `function [x N]=udal(x,m,N)`

Функция `udal` удаляет из массива `x(N)` элемент с номером `m`, функция возвращает модифицированный массив `x` и измененное значение `N`.

В листинге 3.38 приведено содержимое файла *udal_prostoe.m* с комментариями.

```
% Функция udal_prostoe удаляет из массива x(N) простые числа
% и возвращает модифицированный массив x и измененное значение
%  N в качестве результата.
function [x N]=udal_prostoe(x, N)
 i=1;
 while i<=N
% Обращение к функции prostoe для проверки является ли число
% x(i) простым.
      L=prostoe(x(i));
% Число простое (L=1),
      if L==1
% то обращение к функции udal для удаления из массива x(N)
% i-го элемента,
           [x N]=udal(x,i,N);
% иначе переход к следующему элементу массива.
      else
           i=i+1;
      end;
 end;
end % Окончание основной функции udal_prostoe.
```

```
% Функция prostoe проверяет является ли число P простым,
% она возвращает 1, если P – простое,
% 0 – если число P не является простым.
function pr=prostoe(P)
pr=1
for i=2:P/2
 if mod(P,i)==0
      pr=0;
      break;
 end
end
end % Окончание функции prostoe.
% Функция udal удаляет из массива x элемент с номером m.
function [x N]=udal(x,m,N)
% Удаление происходит путем смещения элементов, начиная с m-го
% на одну позицию влево. Выходными элементами функции будут
% массив x, из которого удален один элемент, уменьшенное на 1
% количество(N) элементов в массиве.
for i=m:N-1
 x(i)=x(i+1);
end
% После смещения элементов удаляем последний элемент и
x(:,N)=[];
% уменьшаем количество элементов в массиве на 1.
N=N-1;
end % Окончание функции udal.
```

Листинг 3.38

В листинге 3.38 был использован синтаксис не допускающий вложенность функций друг в друга. Листинг 3.39 содержит текст программы, структура которой допускает вложенность функций.

```
function [x N]=udal_prostoe1(x, N)

function pr=prostoe(P)
pr=1;
for i=2:P/2
 if mod(P,i)==0
      pr=0;
      break;
 end
end
end

function [x N]=udal(x,m,N)
for i=m:N-1
 x(i)=x(i+1);
end
x(:,N)=[];
N=N-1;
end
```

```
i=1;
while i<=N
 L=prostoe(x(i));
 if L==1
      [x N]=udal(x,i,N);
 else
      i=i+1;
 end;
end;
end
```

Листинг 3.39

На листинге 3.40 представлено обращение к функции для удаления простых чисел из массива z(8).

```
>> z=[4 6 8 7 100 13 88 125];
>> [y k]=udal_prostoe(z,8);
>> y
y = 4 6    8    100  88   125
>> k
k= 6
```

Листинг 3.40

Аналогичным образом можно составлять и более сложные функции, в состав которых входит множество вспомогательных функций.

В Octave есть возможность *передавать имя функции как входной параметр*, что существенно расширяет возможности программирования. Вообще говоря, имя функции передается как строка, а ее вычисление осуществляется с помощью функции `feval`.

Функция `feval` предоставляет альтернативный способ вычисления значения функции.

Параметрами функции `feval` являются: строка с именем вызываемой функции, в качестве имени может быть встроенная функция или определенная пользователем функция; параметры этой функции, разделенные запятой.

В качестве параметра можно передать строку с именем функции, а затем с помощью функции `feval` обратиться к передаваемой функции. Рассмотрим передачу функции как параметра на примере решения следующей задачи.

ПРИМЕР 3.18. Вычислить значение функций *sin*(*x*) и *cos*(*x*) в точке $x=\pi/12$.

Вычисление можно осуществить обычным способом или с использованием функции `feval`:

```
>>> x=pi/12
x = 0.26180
>>> sin(pi/13)
ans = 0.23932
>>> feval('sin',x)
ans = 0.25882
>>> cos(pi/13)
ans = 0.97094
>>> feval('cos',x)
ans = 0.96593
```

Листинг 3.41

Как известно, многие функции Octave допускают обращение к ним с различным числом параметров. При этом алгоритм функций анализирует количество входных параметров и осуществляет корректную работу при различном количестве входных параметров.

Рассмотрим, как создавать функции, в которых может использоваться *разное*

количество входных параметров. В качестве входного параметра в этом случае будет использоваться массив ячеек, который позволяет хранить разнородные данные. В этом случае все входные параметры хранятся в виде единственного параметра массива ячеек `varargin`. С помощью функции `length(varargin)` можно вычислить количество поступивших в функцию входных параметров, а с помощью конструкции `varargin{i}` – обратиться к i-му входному параметру.

Рассмотрим простой пример функции с переменным числом параметров.

ПРИМЕР 3.19. Найти сумму всех входных параметров функции. Будем считать, что все входные параметры – скалярные величины.

Выходными параметрами функции будут найденная сумма `sum` и строка `s`, в которой будет храниться аварийное сообщение, если строка не найдена.

В листинге приведена программа решения задачи с комментариями.

```
% Функция вычисления суммы входных параметров.
% Входные параметры хранятся в массиве ячеек.
% Функция возвращает значение суммы в переменной sum,
% а в переменной s формируется сообщение об ошибке,
% если невозможно найти сумму (если среди входных параметров
%  были нечисловые значения).
function [sum s]=sum_var(varargin)
% Переменная pr=1, если все элементы массива ячеек являются
% числами.
pr=1;
% До начала суммирования в переменную sum запишем 0.
sum=0;
% length(varavgin) возвращает количество входных параметров
% в функции sum_var, затем открываем цикл по i для перебора
%  всех входных параметров от 1 до length(varargin).
for i=1:length(varargin)
% С помощью функции isnumeric проверяем, является ли очередной
%  входной параметр числом,
if isnumeric(varargin{i})==1
% если является, добавляем очередной входной
% параметр varargin{i} к sum.
sum=sum+varargin{i};
else
% если не является, записываем в pr=0,
pr=0;
% обнуляем сумму sum
sum=0;
% и прерываем цикл.
break;
end;
end
% Если после проверки всех входных параметров pr=1,
% что означает, что все входные параметры были числами и сумма
% их найдена, очищаем переменную s,где должно хранится
% аварийное сообщение,
if pr==1
s=[];
% иначе записываем в s аварийное сообщение.
```

```
else
s='Во входных параметрах были нечисловые данные'
end
end
>>>% Вызов функции
>>> Summa=sum_var(1,2,3,4,5)
Summa = 15
>>> Summa=sum_var(pi,1.23,e)
Summa = 7.0899
```

Листинг 3.42

Под *рекурсией* в программировании понимается вызов функции из ее тела. Классическими рекурсивными алгоритмами являются возведение числа в целую положительную степень, вычисление факториала и т.д. В рекурсивных алгоритмах функция вызывает саму себя до выполнения какого-либо условия. Рассмотрим пример.

ПРИМЕР 3.20. Вычислить *n*-е число Фибоначчи.

Если нулевой элемент последовательности равен нулю, первый — единице, а каждый последующий представляет собой сумму двух предыдущих, то это последовательность чисел Фибоначчи (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...).

Текст функции:

```
function F=fibonachi(N)
if (N==0) | (N==1)
F=N;
else
F=fibonachi(N-1)+fibonachi(N-2);
end
end
>>>% Вызов функции
>>> fibonachi(2)
ans = 1
>>> fibonachi(0)
ans = 0
>>> fibonachi(6)
ans = 8
```

Листинг 3.43