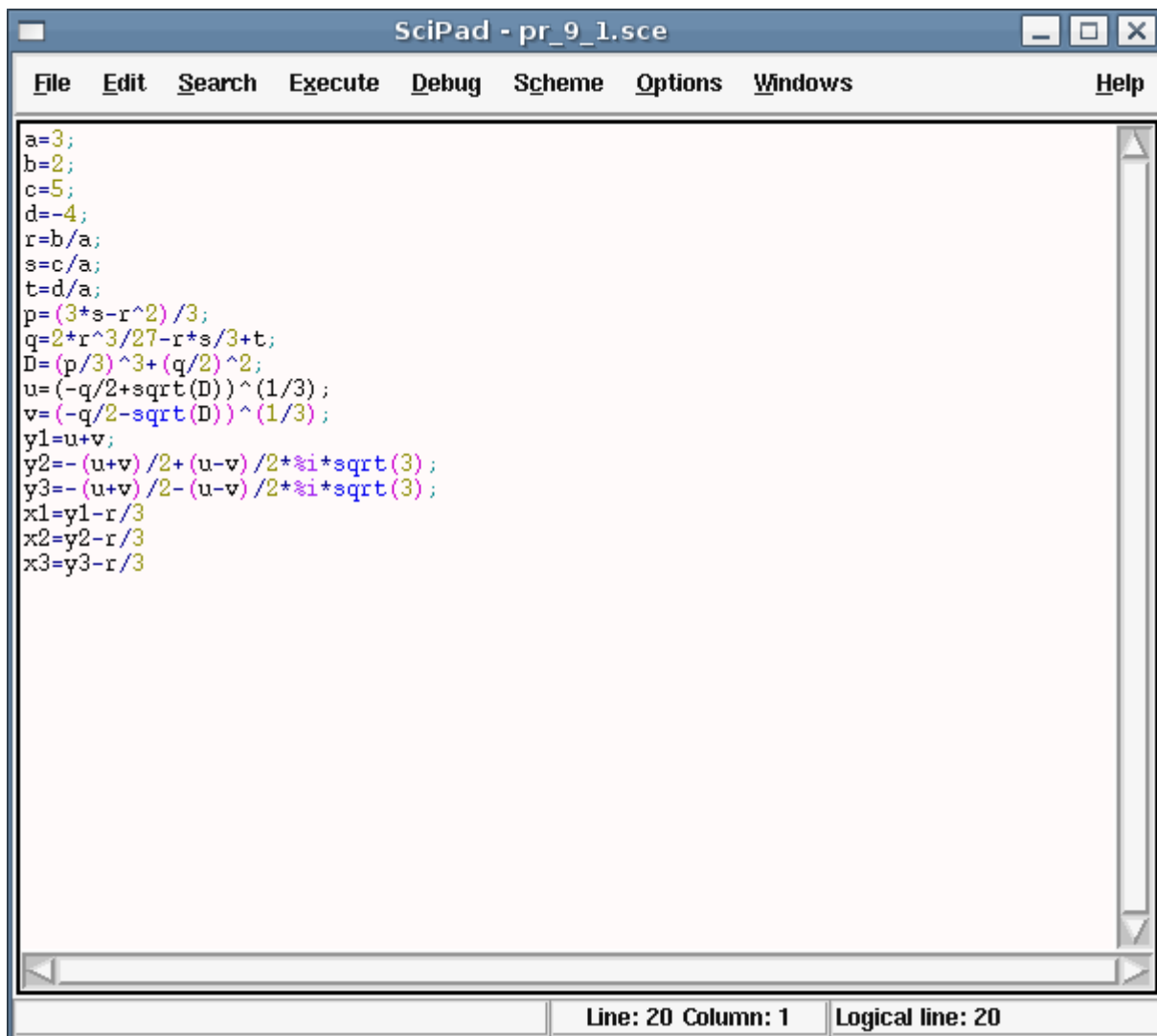


## 9 Программирование в Scilab

В Scilab встроен мощный язык программирования с поддержкой объектов. В этой главе будет описаны возможности структурного программирования, следующая глава будет посвящена визуальному программированию в среде Scilab.

Как уже отмечалось ранее, работа в Scilab может осуществляться в режиме командной строки, но и в так называемом программном режиме. Напомним, что для создания программы (программу в Scilab иногда называют сценарием) необходимо:

1. Вызвать команду **Editor** из меню (см. рис. 9.1).
2. В окне редактора **Scipad** набрать текст программы.



```

SciPad - pr_9_1.sce
File Edit Search Execute Debug Scheme Options Windows Help
a=3;
b=2;
c=5;
d=-4;
r=b/a;
s=c/a;
t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=- (u+v)/2+(u-v)/2*i*sqrt(3);
y3=- (u+v)/2-(u-v)/2*i*sqrt(3);
x1=y1-r/3
x2=y2-r/3
x3=y3-r/3
Line: 20 Column: 1 Logical line: 20

```

Рисунок 9.1. Окно *Scipad* с программой решения кубического уравнения

3. Сохранить текст программы с помощью команды **File Save** в виде файла с расширением **sce**, например **file.sce**.

4. После чего программу можно будет вызвать набрав в командной строке **exec**, например **exec("file.sce")**, или вызвав команду меню **File-Exec...**, или находясь в окне *Scipad* выполнить команду **Execute Load into Scilab(Ctrl+I)**.

Программный режим достаточно удобен, так как он позволяет сохранить разработанный вычислительный алгоритм в виде файла и повторять его при других исходных данных в других сессиях. Кроме обращений к функциям и операторов присваивания, в программных файлах могут использоваться операторы языка программирования Scilab (язык

программирования Scilab будем называть sci-языком).

## 9.1 Основные операторы sci-языка

Изучение sci-языка начнем с функций ввода-вывода.

### 9.1.1 Функции ввода-вывода в Scilab

Для организации простейшего ввода в Scilab можно воспользоваться функциями

```
x=input('title');
```

или

```
x=x_dialog('title', 'stroka');
```

Функция *input* выводит в командной строке Scilab подсказку *title* и ждет пока пользователь введет значение, которое в качестве результата возвращается в переменную *x*. Функция *x\_dialog* выводит на экран диалоговое окно с именем *title*, после чего пользователь может щелкнуть **ОК** и тогда *stroka* вернется в качестве результата в переменную *x*, либо ввести новое значение вместо *stroka*, которое и вернется в качестве результата в переменную *x*. На рисунке 9.2 представлено диалоговое окно, которое формируется строкой `x=x_dialog('Input X', '5')`.

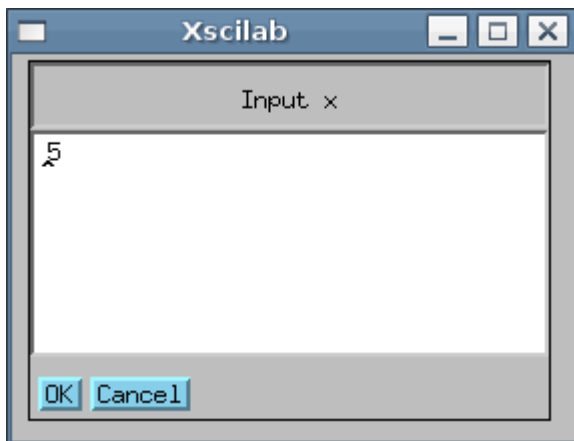


Рисунок 9.2. Окно ввода

Функция *input* преобразовывает введенное значение к числовому типу данных, а функция *x\_dialog* возвращает строковое значение. Поэтому при использовании функции *x\_dialog* для ввода числовых значений, возвращаемую ею строку следует преобразовать в число с помощью функции *evstr*. Поэтому можно предложить следующую форму использования функции *x\_dialog* для ввода числовых значений.

```
x=evstr(x_dialog('title', 'stroka'));
```

Для вывода в текстовом режиме можно использовать функцию *disp* следующей структуры

```
disp(b)
```

Здесь *b* имя переменной или заключенный в кавычки текст .

### 9.1.2 Оператор присваивания

Оператор присваивания имеет следующую структуру

```
a=b
```

здесь *a* имя переменной или элемента массива, *b* значение или выражение. В результате выполнения оператора присваивания переменной *a* присваивается значение выражения *b*.

### 9.1.3 Условный оператор

Одним из основных операторов, реализующим ветвление в большинстве языков программирования, является условный оператор *if*. Существует обычная и расширенная формы оператора *if* в Scilab. Обычный *if* имеет вид

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

Здесь *условие* логическое выражение, *операторы1*, *операторы2* операторы языка Scilab или встроенные функции. Оператор *if* работает по следующему алгоритму: если условие истинно, то выполняются *операторы1*, если ложно *операторы2*.

В Scilab для построения логических выражений могут использоваться условные операторы: *&*, *and* (логическое и), *|*, *or* (логическое или), *~*, *not* (логическое отрицание) и операторы отношения: *<* (меньше), *>* (больше), *=* (равно), *~=*, *<>* (не равно), *<=* (меньше или равно), *>=* (больше или равно).

Зачастую при решении практических задач недостаточно выбора выполнения или невыполнения одного условия. В этом случае можно, конечно, по ветке *else* написать новый оператор *if*, но лучше воспользоваться расширенной формой оператора *if*.

```
if условие1
операторы1
elseif условие2
операторы2
elseif условие 3
операторы3
...
elseif условие n
операторыn
else
операторы
end
```

В этом случае 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 *else* и *elseif* не выполняется, то выполняются операторы по ветке *else*.

#### ЗАДАЧА 9.1.

В качестве примера программирования разветвляющегося процесса рассмотрим решение биквадратного уравнения  $ax^4 + bx^2 + c = 0$ .

Для решения биквадратного уравнения необходимо заменой  $y = x^2$  привести его к квадратному и решить это уравнение. После этого для нахождения корней биквадратного уравнения необходимо будет извлечь корни из найденных значений  $y$ . Входными данными этой задачи являются коэффициенты биквадратного уравнения  $a, b, c$ . Выходными данными являются корни уравнения  $x_1, x_2, x_3, x_4$  или сообщение о том, что действительных корней нет.

Алгоритм состоит из следующих этапов:

1. Ввод коэффициентов уравнения  $a, b$  и  $c$ .
2. Вычисление дискриминанта уравнения  $d$ .
3. Если  $d < 0$ , определяются  $Y_1$  и  $Y_2$ , а иначе вывод сообщения «Корней нет».

4. Если  $y_1 < 0$  и  $y_2 < 0$ , то вывод сообщения «Корней нет».
5. Если  $y_1 \geq 0$  и  $y_2 \geq 0$ , то вычисляются четыре корня по формулам  $\pm\sqrt{y_1}$ ,  $\pm\sqrt{y_2}$  и выводятся значения корней.
6. Если условия 4) и 5) не выполняются, то необходимо проверить знак  $U_1$ .
7. Если  $U_1$  неотрицательно, то вычисляются два корня по формуле  $\pm\sqrt{y_1}$ , иначе оба корня вычисляются по формуле  $\pm\sqrt{y_2}$ .

Программа решения биквадратного уравнения на ссі-языке приведена на листинге 9.1, ее вызов и результаты работы на листинге 9.2.

```
//Ввод значений коэффициентов биквадратного уравнения.
a=input('a=');
b=input('b=');
c=input('c=');
// Вычисляем дискриминант.
d=b*b-4*a*c;
// Если дискриминант отрицателен,
if d<0
// то вывод сообщения,
disp('Real roots are not present');
else
//иначе-вычисление корней соответствующего
// квадратного уравнения.
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
// Если оба корня отрицательны,
if (x1<0)&(x2<0)
// вывод сообщения об отсутствии действительных корней.
disp('Real roots are not present');
// иначе, если оба корня положительны,
elseif (x1>=0)&(x2>=0)
// вычисление четырех корней.
disp('Four real roots');
y1=sqrt(x1);
y2=-y1;
y3=sqrt(x2);
y4=-y2;
disp(y1,y2,y3,y4);
//Иначе,если оба условия (x1<0)&(x2<0) и (x1>=0)&(x2>=0)
// не выполняются,
else
// то вывод сообщения
disp('Two real roots');
// Проверка знака x1.
if x1>=0
//Если x1 положителен, то вычисление двух корней биквадратного
// уравнения, извлечением корня из x1,
y1=sqrt(x1);
y2=-y1;
disp(y1);
```

```

disp(y2);
// иначе (остался один вариант – x2 положителен),
// вычисление двух
// корней биквадратного уравнения извлечением корня из x2.
else
y1=sqrt(x2); y2=-y1;
disp(y1); disp(y2);
end
end end

```

*Листинг 9.1. Программа решения биквадратного уравнения*

```

-->exec('G:/Lecture Scilab EG/2/11.sci');
a-->-6
b-->9
c-->-1
Four real roots
0.3476307
1.1743734
- 0.3476307
0.3476307

```

*Листинг 9.2. Решение биквадратного уравнения*

Найти все корни биквадратного уравнения можно и без оператора *if*, воспользовавшись, тем, что в Scilab определены операции над комплексными числами (см. листинг 9.3).

```

a=input('a=');
b=input('b=');
c=input('c=');
d=b*b-4*a*c;
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
y1=sqrt(x1);
y2=-y1;
y3=sqrt(x2);
y4=-y3;
disp(y1,y2,y3,y4);

```

*Листинг 9.3 Решение биквадратного уравнения*

Результат работы программы, представленной на листинге 9.3 представлены ниже (см. листинг 9.4).

```

-->a=3
-->b=8
-->c=-1
-1.6692213i
1.6692213i
-0.3458800
0.3458800

```

*Листинг 9.4 Комплексные корни биквадратного уравнения*

### 9.1.4 Оператор альтернативного выбора

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

```
select параметр
case значение1 then операторы1
case значение2 then операторы2
...
else операторы
end
```

Оператор *select* работает следующим образом: если значение *параметра* равно *значению1*, то выполняются *операторы1*, иначе если *параметр* равен *значению2*, 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 *else*.

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

Рассмотрим использование оператора *select* на примере решения следующей задачи.

#### ЗАДАЧА 9.2.

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 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 . Вычислить остаток от деления числа *x* на *k* можно по формуле  $x - \text{int}(x/k) * k$ . Следовательно, при построении алгоритма необходимо использовать семь условных операторов.

Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта (см. листинг 9.4). Вызов программы представлен на листинге 9.5.

```
D=input('Enter a number from 1 to 31');
//Вычисление остатка от деления D на 7, сравнение его с
числами
// от 0 до 6.
select D-int(D/7)*7
case 1 then disp('Monday');
case 2 then disp('Tuesday');
case 3 then disp('Wednesday');
case 4 then disp('Thursday');
case 5 then disp('Friday');
case 6 then disp('Saturday');
else
disp('Sunday');
end
```

Листинг 9.4. Решение задачи 9.2

```
-->exec('G:\Lecture Scilab EG\2\l2.sci');disp('exec done');
Enter a number from 1 to 31-->19
Friday
```

### Листинг 9.5.

Рассмотрим операторы цикла в Scilab. В sci-языке есть два вида цикла — оператор цикла с предусловием `while` и оператор `for`.

#### 9.1.5 Оператор `while`

```
Оператор цикла while имеет вид
while условие
операторы
end
```

Здесь *условие* — логическое выражение; *операторы* будут выполняться циклически, пока логическое *условие* истинно.

Оператор цикла *while* обладает значительной гибкостью, но не слишком удобен для организации «строгих» циклов, которые должны быть выполнены заданное число раз. Оператор цикла `for` используется именно в этих случаях.

#### 9.1.6 Оператор `for`

```
Оператор цикла for имеет вид
for x=xn:hx:xk
операторы
end
```

Здесь *x* — имя скалярной переменной — параметра цикла, *xn* — начальное значение параметра цикла, *xk* — конечное значение параметра цикла, *hx* — шаг цикла. Если шаг цикла равен 1, то *hx* можно опустить, и в этом случае оператор *for* будет таким.

```
for x=xn:xk
операторы
end
```

Выполнение цикла начинается с присвоения параметру стартового значения ( $x=xn$ ). Затем следует проверка, не превосходит ли параметр конечное значение ( $x>xk$ ). Если  $x>xk$ , то цикл считается завершенным, и управление передается следующему за телом цикла оператору. Если же  $x \leq xk$ , то выполняются операторы в цикле (тело цикла). Далее параметр цикла увеличивает свое значение на *hx* ( $x=x+hx$ ). После чего снова производится проверка значения параметра цикла, и алгоритм повторяется.

## 9.2 Обработка массивов и матриц в Scilab

Рассмотрим возможности sci-языка для обработки массивов и матриц. Особенностью программирования задач обработки массивов (одномерных, двумерных) на sci-языке является возможность как поэлементной обработки массивов (как в любом языке программирования), так и использование функций Scilab для работы массивами и матрицами.

Рассмотрим основные алгоритмы обработки массивов и матриц и их реализацию на sci-языке.

### 9.2.1 Ввод-вывод массивов и матриц

Ввод массивов и матриц следует организовывать поэлементно, на листингах 9.6, 9.6 приведены программы ввода элементов массивов и матриц на sci-языке.

```
N=input('N=');
disp('Vvod massiva x');
for i=1:N
```

```
x(i)=input('X=');
end
disp(x);
```

*Листинг 9.6. Ввод элементов массива*

```
N=input('N=');
M=input('M=');
disp('Vvod matrici');
for i=1:N
for j=1:M
a(i,j)=input('');
end
end
disp(a);
```

*Листинг 9.7. Ввод элементов матрицы*

Вывод массива (матрицы) можно организовать аналогичным образом в цикле или воспользоваться оператором `disp` для вывода массива (матрицы) целиком<sup>1</sup>.

## 9.2.2 Вычисление суммы и произведения элементов массива (матрицы)

Рассмотрим алгоритм нахождения суммы, который заключается в следующем: вначале сумма равна 0 ( $s=0$ ), затем к  $s$  добавляем первый элемент массива и результат записываем опять в переменную  $s$ , далее к переменной  $s$  добавляем второй элемент массива и результат записываем в  $s$  и далее аналогично добавляем к  $s$  остальные элементы массива. При нахождении суммы элементов матрицы последовательно суммируем элементы всех строк.

Алгоритм нахождения произведения следующий: на первом начальное значение произведения равно 1 ( $p=1$ ), затем последовательно умножаем  $p$  на очередной элемент, и результат записываем в  $p$  и т.д. На листингах 9.8 - 9.11 представлены элементы программ, реализующие эти алгоритмы.

```
//Записываем в переменную s число 0.
s=0;
//Перебираем все элементы массива
for i=1:length(x)
//накопление суммы
s=s+x(i);
end
```

*Листинг 9.8. Программа вычисления суммы элементов массива*

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

*Листинг 9.9. Программа вычисления произведения элементов массива*

```
s=0;
```

---

<sup>1</sup> Напоминаем читателю, что если после имени массива (матрицы) не поставить точку с запятой, то массив (матрица) будет выведена в окне Scilab.



```
//Вычисляем количество строк N и столбцов M матрицы A.
[N,M]=size(A);
for i=1:N
for j=1:M
s=s+a(i,j);
end
end
disp(s);
```

*Листинг 9.10. Программа вычисления суммы элементов матрицы*

```
//Начальное значение произведение (p) равно 1.
p=1;
//Вычисляем количество строк N и столбцов M матрицы A.
[N,M]=size(A);
//Перебираем все строки матрицы.
for i=1:N
//Перебираем все столбцы матрицы.
for j=1:M
Умножаем значение p на текущий элемент матрицы.
p=p*a(i,j);
end
end
```

*Листинг 9.11. Программа вычисления произведения элементов матрицы*

### 9.2.3 Поиск максимального (минимального) элемента массива (матрицы)

Алгоритм решения задачи поиска максимума и его номера в массиве следующий. Пусть в переменной *s* 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 (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 9.12 представлен фрагмент программы поиска максимума.

```
//Записываем в Max значение первого элемента массива.
Max=a(1);
//Записываем в Nmax значение номер максимального элемента
//массива, сейчас это число 1.
Nmax=1;
//Перебираем все элементы массива, начиная со второго.
for i=2:N
//Если текущий элемент массива больше Max,
if x(i)>Max
//то текущий элемент массива объявляем максимальным,
Max=x(i);
//а его номер равен i.
Nmax=i;
end;
```

end;

*Листинг 9.12. Реализация алгоритма поиска максимума*

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

На листинге 9.13 представлена программа поиска минимального элемента матрицы и его индексов: *Nmin* номер строки, *Lmin* номер столбца минимального элемента.

Обратите внимание, что при поиске минимального (максимального) элемента матрицы циклы по *i* и *j* начинаются с 1. Если написать с двух, то при обработке элементов будет пропущена первая строка или первый столбец при сравнении  $a_{i,j}$  с *min*.

```
//Записываем в Min a(1,1), в Nmin и Lmin число 1.
Min=a(1,1); Nmin=1; Lmin=1;
for i=1:N
for j=1:M
//Если текущий элемент матрицы меньше Min,
if a(i,j)<Min
//то текущий элемент массива объявляем минимальным,
Min=a(i,j);
//а его индексы равны i и j.
Nmin=i;
Lmin=j;
end;
end;
end;
```

*Листинг 9.13. Программа поиска минимального элемента матрицы и его индексов*

## 9.2.4 Сортировка элементов массива

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив *X* из *n* элементов будет отсортирован<sup>2</sup> в порядке возрастания значений его элементов, если

$$X[1] \leq X[2] \leq \dots \leq X[n],$$

и в порядке убывания, если

$$X[1] \geq X[2] \geq \dots \geq X[n].$$

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, ..., (*i* - 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, далее следует повторить алгоритм до тех пор, пока не упорядочим массив.

Алгоритм сортировки по убыванию будет отличаться от приведенного заменой знака  $>$  на  $<$ . На листинге 9.14 представлен фрагмент программы на *sci*-языке, реализующий алгоритм сортировки по возрастанию.

```
x=[-3 5 7 49 -8 11 -5 32 -11];
for i=1:length(x)-1
```

<sup>2</sup> Зачастую алгоритм сортировки называют алгоритмом упорядочивания элементов массива.

```

for j=1:length(x)-1
//Сравниваем два соседних элемента, и если предыдущий меньше
// последующего,
if x(j)>x(j+1)
//то меняем их местами.
b=x(j);
x(j)=x(j+1);
x(j+1)=b;
end;
end;
end;
//Вывод упорядоченного массива.
disp(x);

```

Листинг 9.14. Программа упорядочивания по возрастанию

### 9.2.5 Удаление элемента из массива

Необходимо удалить из массива  $x$ , состоящего из  $n$  элементов,  $m$  й по номеру элемент. Для этого достаточно записать  $(m+1)$ -й элемент на место элемента с номером  $m$ ,  $(m+2)$ -й на место  $(m+1)$ -го, ...,  $n$ -й на место  $(n-1)$ -го, после чего удалить последний  $n$ -й элемент. На листинге 9.15 приведен фрагмент программы, реализующий описанный алгоритм.

```

x=[3 2 1 5 4 6 8 7];
disp(x);
n=length(x);
//Ввод номера удаляемого элемента.
m=input('m=');
//Сдвиг всех элементов, начиная с m-го на один влево.
for i=m:n-1
x(i)=x(i+1);
end;
// Удаление n-го элемента из массива.
x(:,n)=[];
//Уменьшение n на 1.
n=n-1;
//Вывод преобразованного массива.
disp(x);

```

Листинг 9.15. Программа удаления  $m$ -го элемента из массива  $x(n)$

## 9.3 Работа с файлами в Scilab

Рассмотрим функции Scilab для работы с файлами<sup>3</sup>.

### 9.3.1 Функция открытия файла *mopen*

Как и в любом другом языке программирования работа с файлом начинается с его открытия. Для открытия файла в *sci* языке предназначена функция *mopen*, которая имеет вид:

```
[fd,err]=mopen(file, mode)
```

*file* строка ,в которой хранится имя файла ,

<sup>3</sup> Авторы обращают внимание читателей, что функции работы с файлами в *sci*-языке аналогичны соответствующим функциям в языке C.

*mode* режим работы с файлом :

- 'r' текстовый файл открывается в режиме чтения ,
- 'rb' двоичный файл открывается в режиме чтения ,
- 'w' открывается пустой текстовый файл , который предназначен только для записи информации;
- 'wb' открывается пустой двоичный файл , который предназначен только для записи информации;
- 'a' открывается текстовый файл , который будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- 'ab' открывается двоичный файл , который будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- 'r+' открывается текстовый файл , который будет использоваться в режиме чтения и записи;
- 'rb+' открывается двоичный файл , который будет использоваться в режиме чтения и записи;
- 'w+' создаваемый пустой текстовый файл предназначен для чтения и записи информации;
- 'wb+' создаваемый пустой двоичный файл предназначен для чтения и записи информации;
- 'a+' открываемый текстовый файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан;
- 'ab+' открываемый двоичный файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан.

Функция  *fopen*  возвращает идентификатор открытого файла  *fd*  и код ошибки  *err* . Идентификатор файла  *имя (код )*, по которому описанные ниже функции будут обращаться к реальному файлу на диску. В переменной  *err*  возвращается значение 0, в случае удачного открытия файла. Если  *err ≠ 0*  , то файл открыть не удалось.

### 9.3.2 Функция записи в текстовый файла *fprintf*

Функция записи в текстовый файл  *fprintf*  имеет вид  
 *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][модификатор]тип<sup>4</sup>**

Значения параметров строки преобразования приведены в таблице 9.1.

Таблица 9.1. Значение параметров строки преобразования

Параметр	Назначение
<b>Флаг</b>	
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «-»
Пробел	Перед положительным числом выводится пробел, перед отрицательным «-»

4 Параметры в квадратных скобках являются необязательными и могут отсутствовать.

Параметр	Назначение
#	Выводится код системы счисления: 0 перед восьмеричным числом, 0x (0X) перед шестнадцатеричным числом.
<b>Ширина</b>	
n	Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции заполняются пробелами.
0n	То же, что и n, но незаполненные позиции заполняются нулями.
<b>Точность</b>	
ничего	Точность по умолчанию
n	Для типов e, E, f выводить n знаков после десятичной точки
<b>Тип</b>	
c	При вводе символьный тип char, при выводе один байт.
d,i	Десятичное со знаком
i	Десятичное со знаком
o	Восьмеричное int unsigned
u	Десятичное без знака
x, X	Шестнадцатеричное int unsigned, при x используются символы a-f, при X A-F.
f	Значение со знаком вида [-]dddd.dddd
e	Значение со знаком вида [-]d.ddddE[+ -]ddd
E	Значение со знаком вида [-]d.ddddE[+ -]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
<b>Модификатор (типа)</b>	
h	Для d, i, o, u, x, X короткое целое
l	Для d, i, o, u, x, X длинное целое

В строке вывода могут использоваться некоторые специальные символы, приведенные в табл. 9.2.

Таблица 9.2. Некоторые 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 ?

### 9.3.3 Функция чтения данных из текстового файла `fscanf`

При считывании данных из файла можно воспользоваться функцией `mfscanf` следующего вида

```
A=mfscanf(f, s1)
```

Здесь  $f$  – идентификатор файла, который возвращается функцией `mopen`,  $s1$  – строка форматов вида

```
%[ширина][.точность]тип
```

Функция `mfscanf` работает следующим образом: из файла с идентификатором  $f$  считываются в переменную  $A$  значения 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.

### 9.3.4 Функция закрытия файла `fclose`

После выполнения всех операций с файлом он должен быть закрыт с помощью функции `fclose` следующей структуры

```
fclose(f)
```

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

Пример создания текстового файла приведен на листинге 9.16.

```
//В текстовом файле abc.txt хранятся размеры матрицы N и M,
```

```
// и сама матрица A(N,M)
```

```
//N – количество строк матрицы.
```

```
N=3;
```

```
//M- количество столбцов матрицы.
```

```
M=4;
```

```
A=[2 4 6 7; 6 3 2 1; 11 12 34 10];
```

```
//Открываем пустой файл abc.txt в режиме записи.
f=fopen('abc.txt','w');
//Записываем в файл abc.txt N и M, разделенные символом
//табуляции.
fprintf(f,'%d\t%d\n',N,M);
for i=1:N
for j=1:M
//Записываем в файл abc.txt очередной элемент матрицы A.
fprintf(f,'%g\t',A(i,j));
end
//По окончании записи очередной строки, записываем в файл
//символ «конец строки».
fprintf(f,'\n');
end
fclose(f);
```

*Листинг 9.16. Создание текстового файла*

Созданный текстовый файл можно увидеть на рис. 9.3.

Программа чтения данных из этого текстового файла приведена на листинге 9.17.

```
f=fopen('abc.txt','r');
N=mfscanf(f,'%d');
M=mfscanf(f,'%d');
for i=1:N
for j=1:M
A(i,j)=mfscanf(f,'%g');
end
end
fclose(f);
```

*Листинг 9.17. Чтение из текстового файла*

Результаты работы файла-сценария, представленного на листинг 9.16 можно увидеть на листинге 9.18.

```
N =
3.
M =
4.
A =
! 2. 4. 6. 7. !
! 6. 3. 2. 1. !
! 11. 12. 34. 10. !
```

*Листинг 9.18. Результат работы файла сценария*

## **9.4 Пример программы в Scilab**

В качестве примера программы на sci-языке рассмотрим следующую задачу.

### **ЗАДАЧА 9.3.**

Положительные числа из массива Y переписать в массив X, удалить из массива X элементы, меньшие среднего арифметического и расположенные после минимального элемента.

Программа решения задачи представлена на листинге 9.19. Использование переменной *min1* в программе обусловлено тем, что в Scilab есть встроенная функция *min*. Использование переменной с именем *min* не позволит использовать встроенную функцию *min*.

```
//Ввод количества элементов в массиве y.
N=input('N=');
disp('Vvod massiva Y');
//Цикл для ввода элементов в массиве y.
for i=1:N
y(i)=input('Y=');
end
disp(y);
//Переменная k содержит количество положительных элементов в
//массиве y, и как следствие количество элементов в массиве x.
//Первоначально k=0.
k=0;
//Перебираем все элементы в массиве y.
for i=1:N
//Если текущий элемент положителен, то
if y(i)>0
//значение переменной k увеличиваем на 1,
k=k+1;
//а элемент массива y переписываем в x.
x(k)=y(i);
```



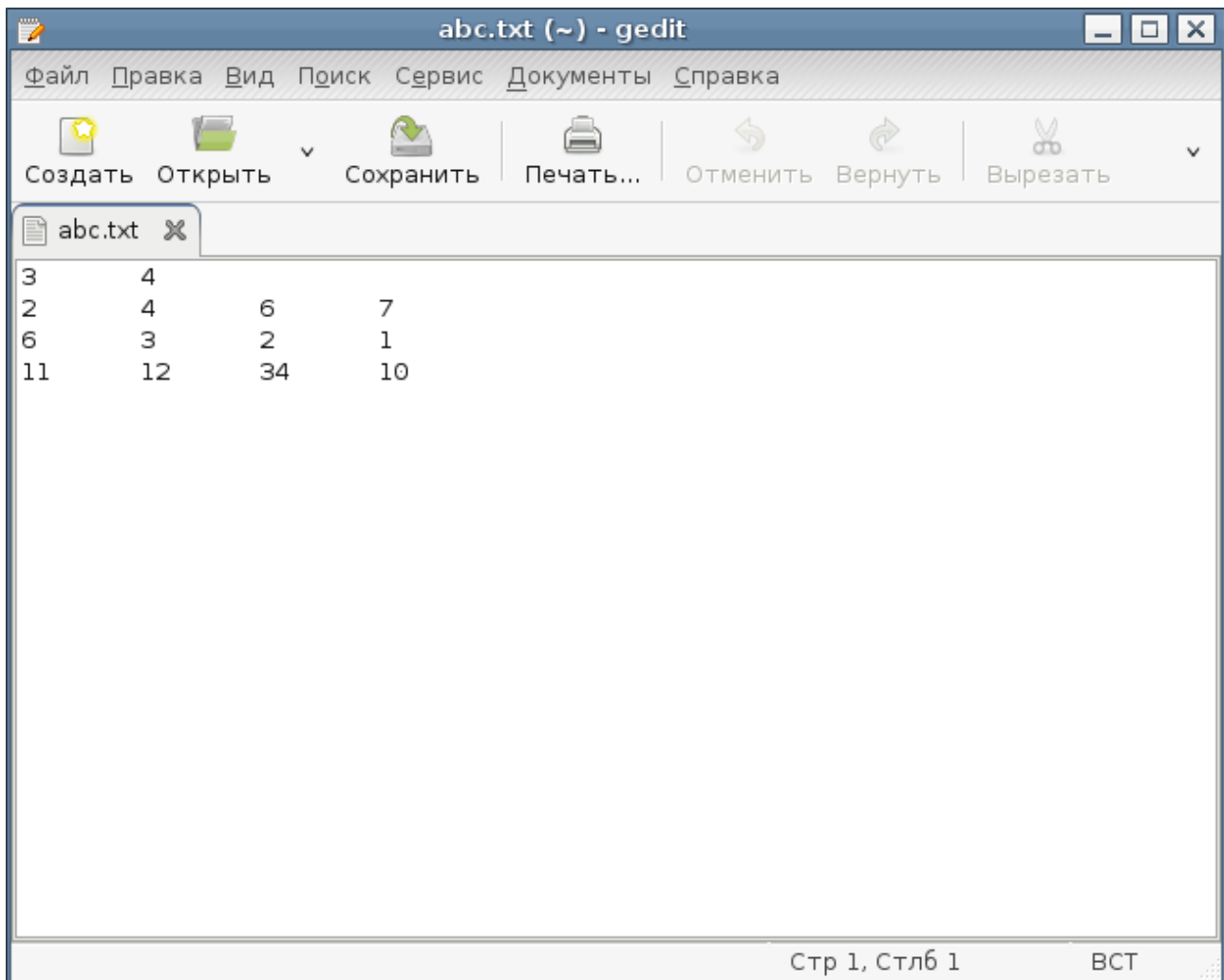


Рисунок 9.3. Созданный текстовый файл

```

end;
end;
//вывод массива x.
disp(x);
//В переменной s будем хранить сумму элементов массива x, в
// переменной min1 – минимальное значение в массиве x, в Nmin
// - номер минимального элемента в массиве x..
s=x(1);
min1=x(1);
Nmin=1;
//Цикл для поиска минимума и суммы среди элементов массива x.
for i=2:k
//Накапливание суммы.
s=s+x(i);
//Поиск минимума.
if x(i)<min1
min1=x(i);
Nmin=i;
end;
end;

```

```

//Начиная с минимального, перебираем все элементы массива, и
//если
i=Nmin;
while i<=k
//текущий элемент меньше среднего арифметического,
if x(i)<s/k
//то удаляем текущий элемент. Обратите внимание, при удалении
// происходит сдвиг элементов на 1 влево, и поэтому не надо
// увеличивать i на 1, и переходить к следующему элементу
после
// удаления.
for j=i:k-1
x(j)=x(j+1);
end;
//Уменьшение количества элементов в массиве на 1.
x(k)=[];
k=k-1;

else
//Если удаление не происходило, то переходим к следующему
// элементу массива.
i=i+1;
end;
end;
disp(x);

```

Листинг 9.19. Решение задачи 9.3.

## 9.5 Функции в Scilab

В Scilab несложно оформлять собственные функции. Структура функции в Scilab следующая:

```

function [y1,y2,...,yn]=ff(x1,x2,...,xm)
операторы
endfunction

```

Здесь  $x1, x2, \dots, xm$  список входных параметров функции;  $y1, y2, \dots, yn$  список выходных параметров функции,  $ff$  имя функции.

Если вызываемая функция находится не в текущем файле, то перед ее вызовом следует загрузить файл, в котором находится функция с помощью функции `ехес` следующей структуры.

```
ехес('file', -1).
```

Здесь *file* имя файла на диске, в котором находится вызываемая функция.

Рассмотрим пример с использованием функций и файлов.

### ЗАДАЧА 9.4.

В матрице  $A(N,N)$  найти сумму элементов, расположенных на диагоналях матрицы, вычислить количество элементов, равных максимальному элементу матрицы. Число  $N$  и матрица  $A$  хранятся в текстовом файле **primer.txt** (см. рис. 9.20).

Для нахождения суммы (*summa*), максимума (*maximum*) и количества максимумов (*kolichestvo*) оформим функцию, заголовок ее будет иметь вид:

```
function [summa, maximum, kolichestvo]=matrica_A(N,N)
```

Текст функции приведен на листинге 9.20.

```

function [summa, maximum, kolichestvo]=matrica_A(A,N)
summa=0;
for i=1:N
//Суммируем элементы, расположенные на главной A(i,i) и
// побочной A(i,N+1-i) диагоналях
summa=summa+A(i,i)+A(i,N+1-i);
end
//Если количество строк в матрице нечетное, то есть элемент,
// который расположен одновременно на главной и побочной
// диагоналях, мы его просуммировали дважды, и как элемент на
// главной, и как элемент побочной диагонали, поэтому его надо
// вычесть из суммы.
if (N-int(N/2)*2)==1
summa=summa-A(int(N/2)*2+1,int(N/2)*2+1);
end
//Элемент A(1,1) объявляем максимальным, количество максимумов
// равно 1.
maximum=A(1,1);kolichestvo=1;
for i=1:N
for j=1:N
//Если текущий элемент A(i,j) больше максимального, то его и
// объявляем максимальным, количество максимумов равно 1.
if A(i,j)>maximum
maximum=A(i,j);
kolichestvo=1;
//Если текущий элемент равен максимальному, то количество
// максимумов увеличиваем на 1.
elseif A(i,j)==maximum
kolichestvo=kolichestvo+1;
end
end
end
endfunction

```

*Листинг 9.20. Текст функции matrica\_A*

Текст функции, предназначенной для чтения матрицы из файла и вызова функции matrica\_A приведен на листинге 9.21, результаты на листинге 9.22.

```

f=mopen('G:\primer.txt','r');
N=mfscanf(f,'%d');
for i=1:N
for j=1:N
B(i,j)=mfscanf(f,'%g');
end
end
mclose(f);
[s,m,k]=matrica_A(B,N);

```

*Листинг 9.21. Вызов функции matrica\_A*

```

-->exec("matrica_2.sci");

```

```
-->s  
s =  
21.  
-->m  
m =  
67.  
-->k  
k =  
4.
```

*Листинг 9.22. Результаты работы программы*

В этой части были рассмотрены основные возможности программирования в Scilab. В следующей главе будут рассмотрены возможности построения визуальных программ.