9. Программирование в Scilab

Как уже рассматривалось раннее, 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.

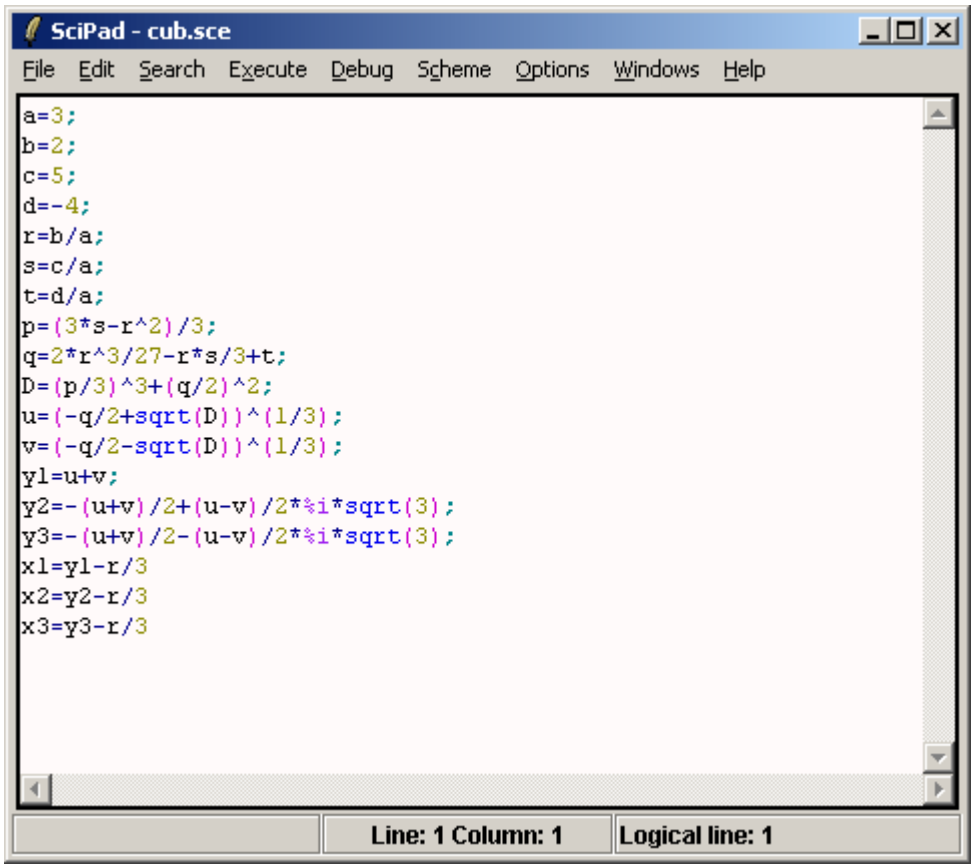


Рис. 9.1. Окно Scipad с текстом программы

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...** .

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

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 (см. рис. 9.2), после чего пользователь может щелкнуть OK 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**.

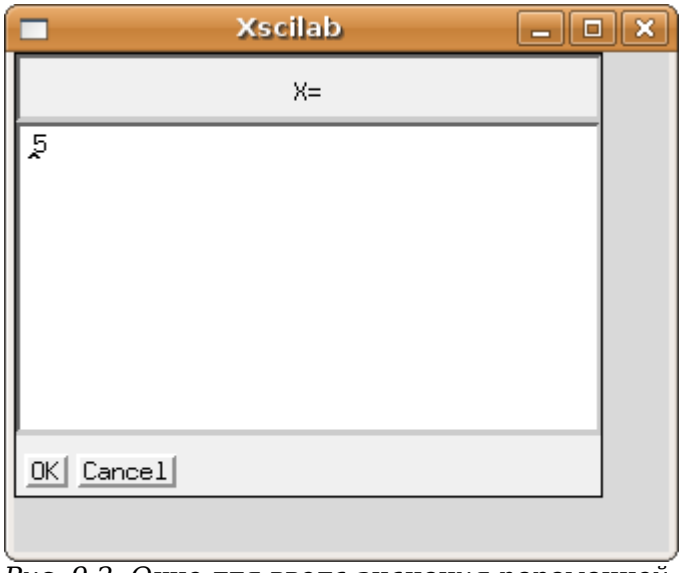


Рис. 9.2. Окно для ввода значения переменной

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 следующем алгоритму: если условие истинно, то выполняются **операторы1**, если ложно – **операторы2** (см. рис. 9.3).

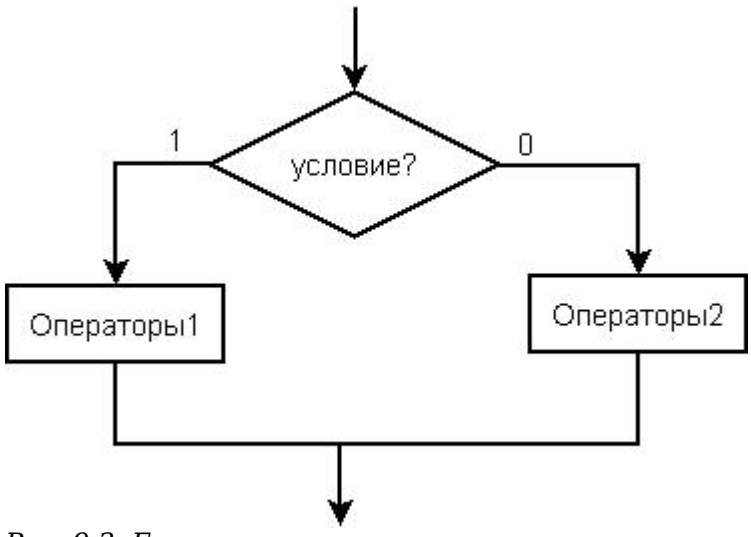


Рис. 9.3. Блок-схема условного оператора

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** (см. рис. 9.4).

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

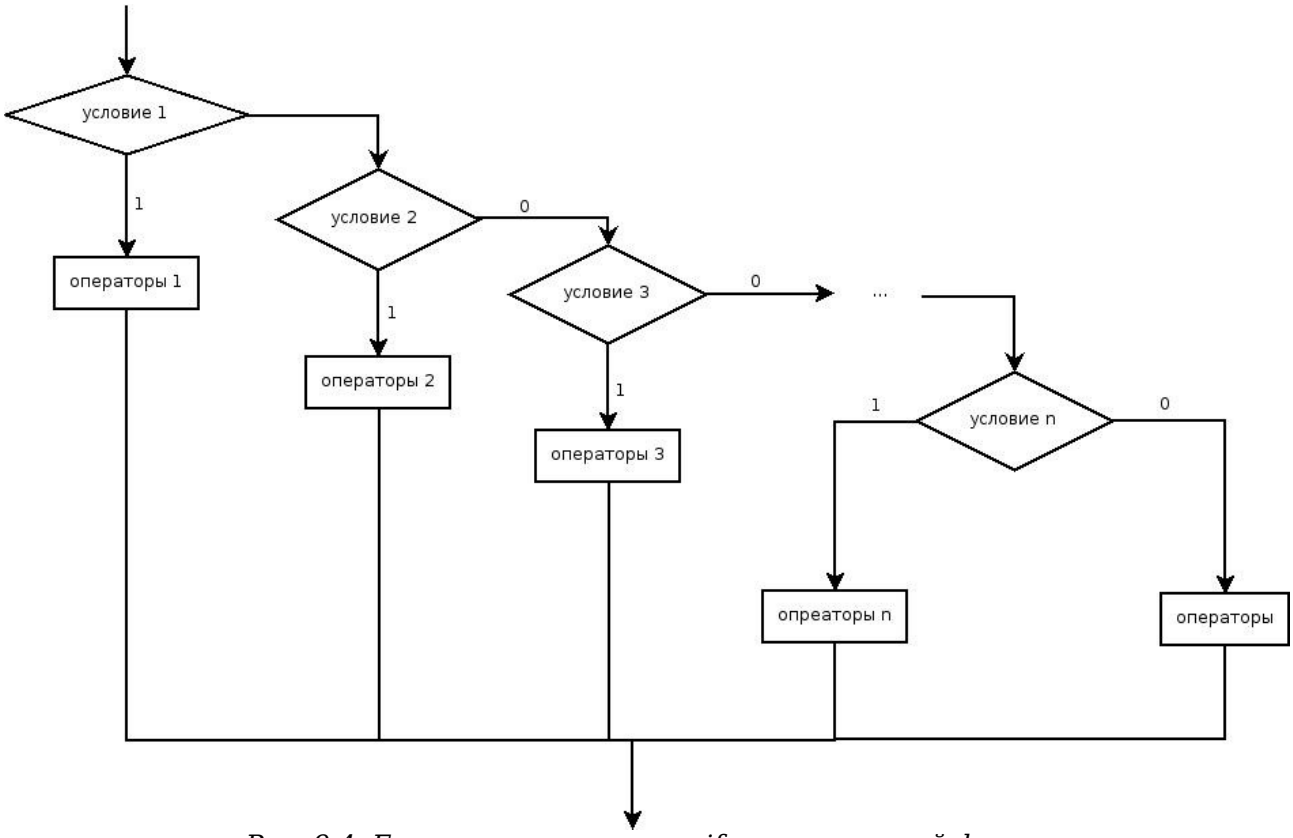


Рис. 9.4. Блок-схема оператора if в расширенной форме

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**. На рис. 4. представлена блок-схема

расширенной формы оператора **if**.

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. Входными данными этой задачи являются коэффициенты биквадратного уравнения a, b, c. Выходными данными являются корни уравнения x_1, x_2, x_3, x_4.

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 корней нет.
4. Если y_1, $y_2< 0$, то корней нет.
5. Если y_1, $y_2<0$, то вычисляются четыре корня по формулам и выводятся значения корней.
6. Если условия 4) и 5) не выполняются, то необходимо проверить знак y1.
7. Если y1 неотрицательно, то вычисляются два корня по формуле ±sqrt(y_1), иначе оба корня вычисляются по формуле ±sqrt(y_2) Вычисленные значения корней выводятся.

Блок-схема решения этой задачи представлена на рис. 9.5, программа решения биквадратного уравнения на sci-языке приведена на листинге 9.1, ее вызов и результаты работы на листинге 9.2.

```
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.

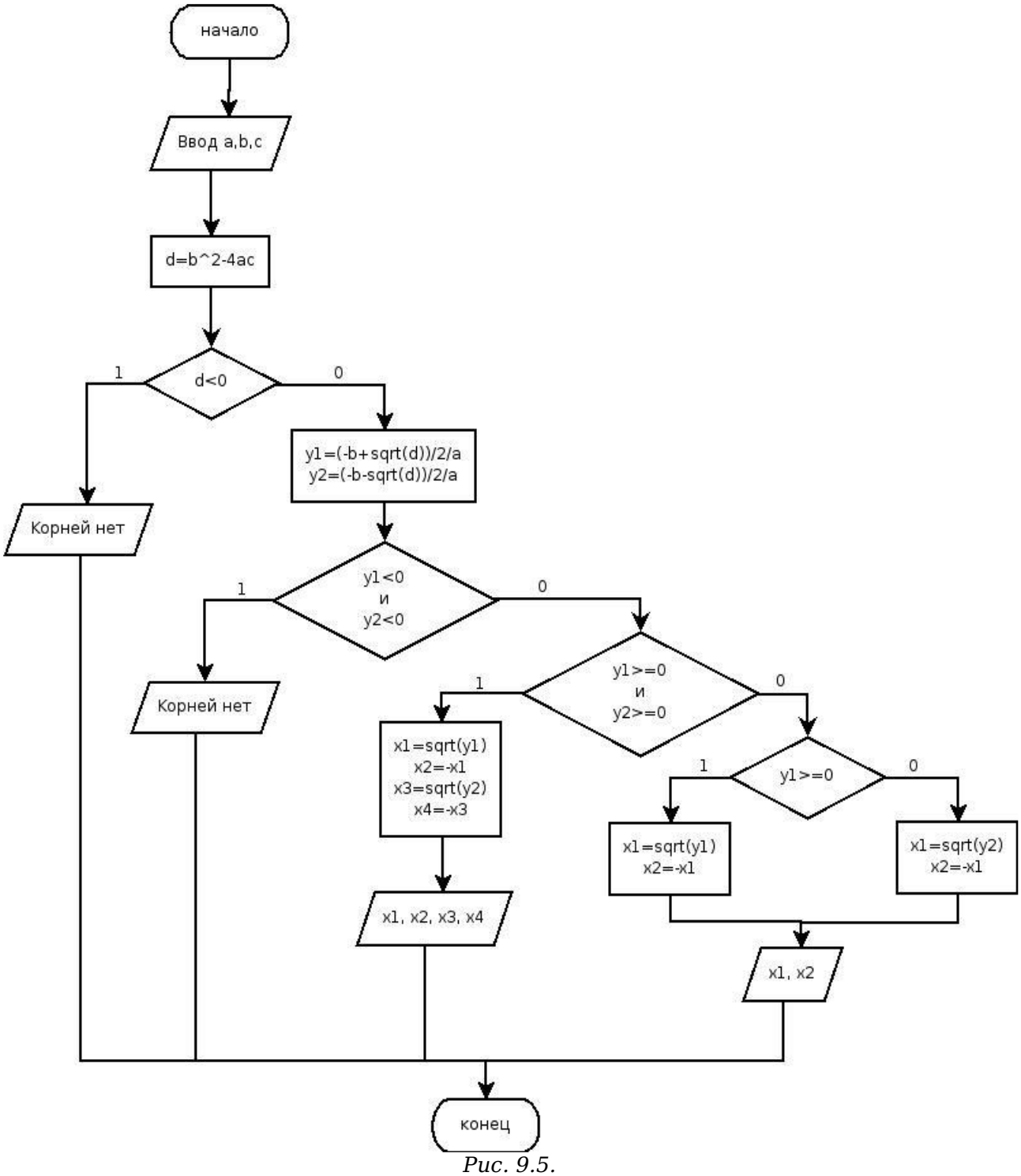


Рис. 9.5.

```
-->exec("G:/Lecture Scilab EG/2/l1.sci");
a=-->-6
b=-->9
c=-->-1
 Four real roots
    0.3476307
```

```
    1.1743734
  - 0.3476307
    0.3476307
```

Листинг 9.2.

9.1.4. Оператор альтернативного выбора

Еще одним способом организации разветвлений является оператор 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 **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 **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 вычисляем остаток от деления числа x на k

x-int(x/k)*x

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 (см. листинг 9.3). Вызов программы представлен на листинге 9.4.

```
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 9.3.

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

Листинг 9.4.

Рассмотрим операторы цикла в Scilab. В Sci-языке Scilab 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. Блок-схема оператора while представлена на рис. 9.6.

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.

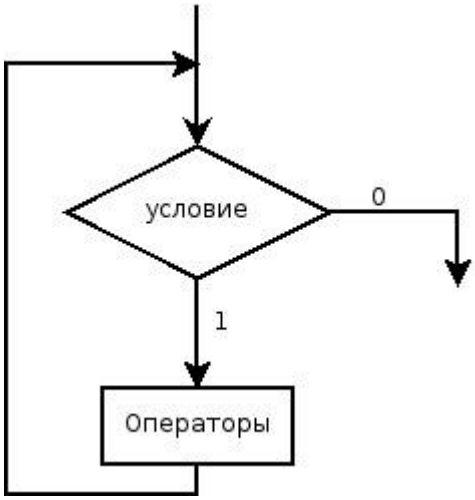


Рис. 9.6. Блок-схема оператора while

9.1.6. Оператор for

Оператор цикла **for** имеет вид

for x=xn:hx:xk

операторы

end

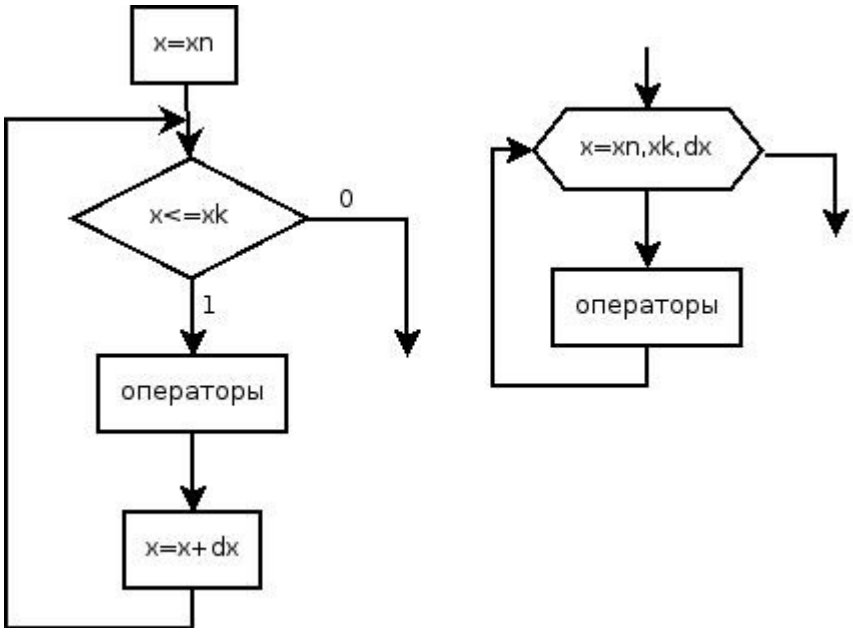
Здесь **x** – имя скалярной переменной – параметра цикла, **xn** – начальное значение параметра цикла, **xk** – конечное значение параметра цикла, **hx** – шаг цикла. Если шаг цикла равен 1, то **hx** можно опустить, и в этом случае оператор **for** будет таким.

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 результат проверки утвердительный, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняются операторы в цикле (тело цикла). Далее параметр меняет свое значение (x=x+hx). Далее снова производится проверка значения параметра цикла, и алгоритм повторяется (см. рис. 9.7).



а) *б)*

Рис. 9.7.Оператор for:
-а) блок-схема алгоритма оператора,
б) стандартная блок-схема оператора

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. Обработка массивов и матриц в Scilab

9.2.1. Ввод-вывод массивов и матриц

Ввод массивов и матриц следует организовывать поэлементно, на рис. 9.8 приведена блок-схема алгоритма ввода элементов массивов, а на рис. 9.9 – матриц. На листингах 9.5, 9.6 приведены программы ввода элементов массивов и матриц на М-языке, реализующие эти алгоритмы.

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

Листинг 9.5. 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 9.6. Ввод элементов матрицы

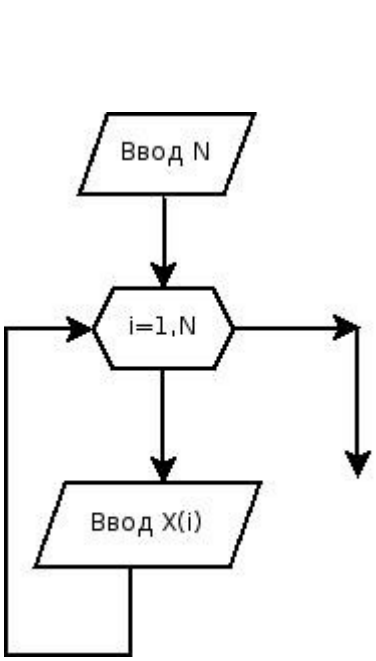


Рис. 9.8.Блок-схема ввода элементов массива

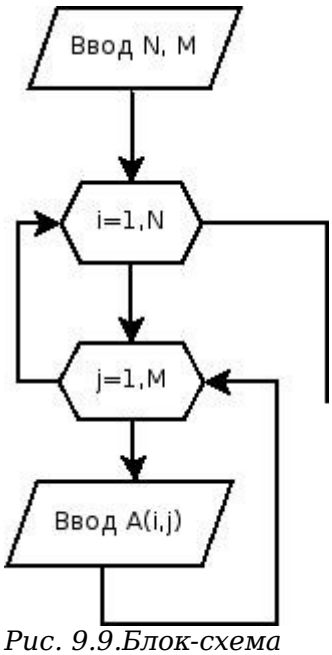


Рис. 9.9.Блок-схема ввода элементов матрицы

9.2.2. Вычисление суммы и произведения элементов массива (матрицы)

Алгоритм нахождения суммы состоит 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.

На рис. 9.10, 9.11 приведены блок-схемы алгоритмов нахождения суммы и произведения элементов массива, на рис 9.12, 9.13 – алгоритмы нахождения суммы и произведения элементов матрицы. На листингах 9.7-9.10 представлены элементы программ, реализующие эти алгоритмы.

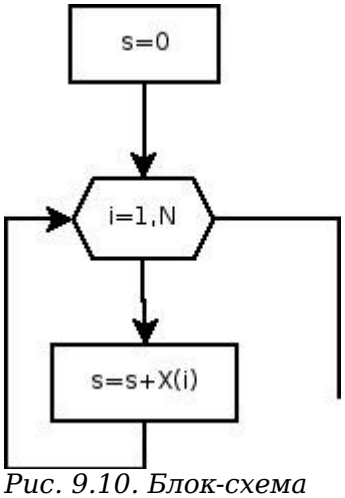


Рис. 9.10. Блок-схема алгоритма нахождения суммы элементов массива

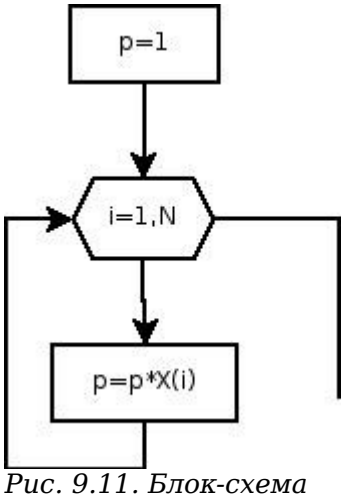


Рис. 9.11. Блок-схема алгоритма нахождения элементов массива

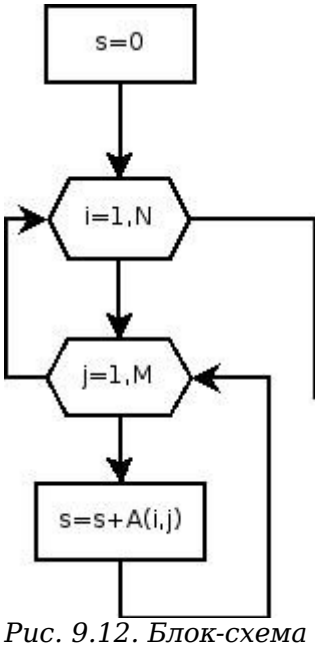


Рис. 9.12. Блок-схема нахождения суммы элементов матрицы

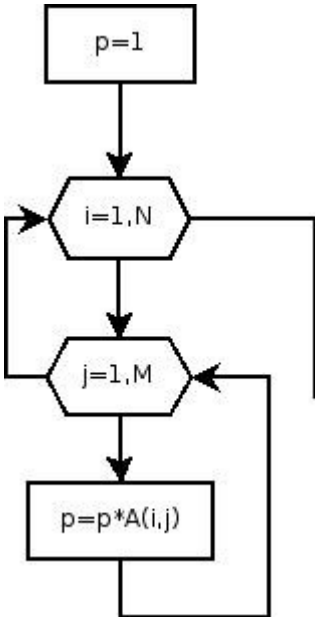


Рис. 9.13.Блок-схема нахождения произведения элементов матрицы

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

Листинг 9.7. Программа вычисления суммы элементов массива

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

Листинг 9.8. Программа вычисления произведения 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);
s=0;
for i=1:N
for j=1:M
s=s+a(i,j);
end
end
disp(s);
```

Листинг 9.9. Программа вычисления суммы элементов матрицы

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

Листинг 9.10. Программа вычисления произведения элементов матрицы

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

Пусть в переменной с именем Max хранится значение максимального элемента массива, а в переменной с именем Nmax – его номер. Алгоритм решения задачи поиска максимума и его номера в массиве следующий. 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. Алгоритм нахождения максимального элемента в массиве приведен в блок-схеме на рис. 9.14.

На листинге 9.11 представлен фрагмент программы поиска максимума.

```
Max=a(1);
Nmax=1;
for i=2:N
if x(i)>Max
Max=x(i);
Nmax=i;
end;
end;
```

Листинг 9.11. Реализация алгоритма поиска максимума

Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в условном блоке и, соответственно, в конструкции if текста программы знак поменяется с > на <. На рис. 9.15 представлена блок-схема 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. Иначе при обработке элементов будет пропущена первая строка или первый столбец при сравнении $a_{i,j}$ с min. На листинге 9.12 представлена реализация этого алгоритма.

```
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 9.12. Программа поиска минимального элемента матрицы и его индексов

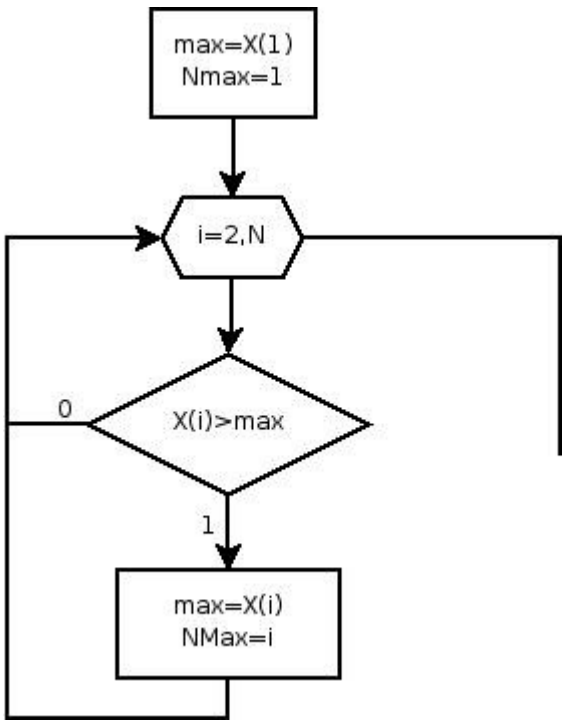


Рис. 9.14.Поиск минимального элемента в массиве и его номера

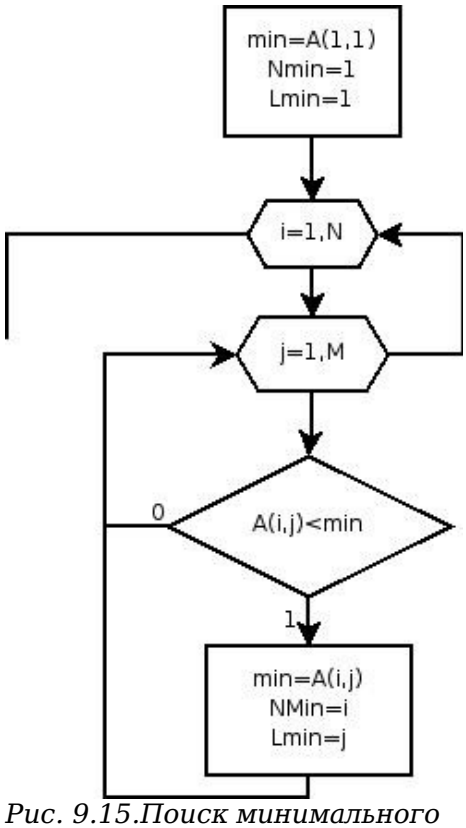


Рис. 9.15.Поиск минимального элемента матрицы и его индексов

9.2.4. Алгоритм упорядочивания элементов массива

Сортировка (упорядочивание) представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив X из n элементов будет отсортирован в порядке возрастания значений его элементов, если

$X[1] \le X[2] \le \ldots \le X[n]$,

и в порядке убывания, если

$X[1] \ge X[2] \ge \ldots \ge X[n]$.

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, (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. Блок-схема алгоритма представлена на рис. 9.16.

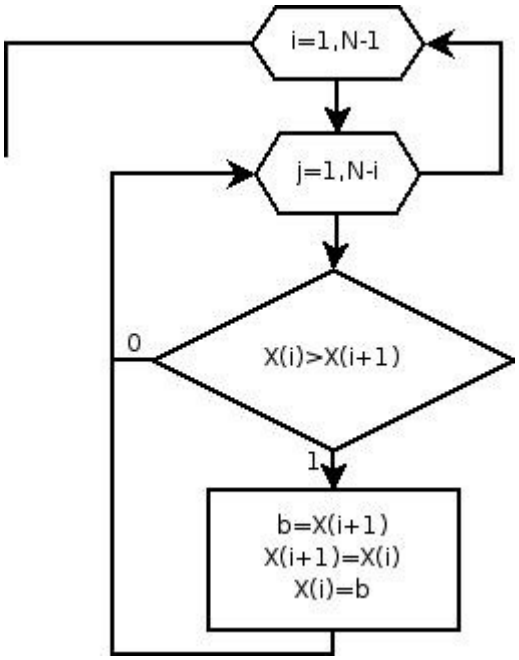


Рис. 9.16. Алгоритм упорядочивания массива по возрастанию

Алгоритм сортировки по убыванию будет отличаться от приведенного на рис. 9.16 заменой знака > на <. На листинге 13 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
for j=1:length(x)-i
if x(j)>x(j+1)
b=x(j);
x(j)=x(j+1);
x(j+1)=b;
end;
end;
end;
disp(x);
```

Листинг 9.13. 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 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.

Алгоритм удаления из массива x размерностью n элемента с номером m приведен на рис. 9.17. На листинге 9.14 приведен фрагмент программы, реализующий этот алгоритм.

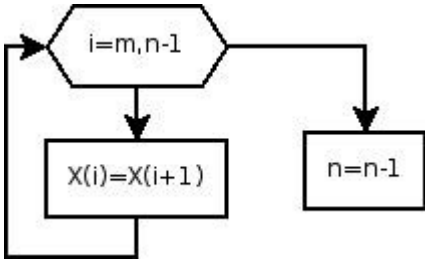


Рис. 9.17. Алгоритм удаления

```
x=[3 2 1 5 4 6 8 7];
disp(x);
n=length(x);
m=input('m=');
for i=m:n-1
x(i)=x(i+1);
end;
// Удаление n-го элемента из массива.
x(:,n)=[];
n=n-1;
disp(x);
```

Листинг 9.14. Программа удаления m-го элемента из массива x(n)

9.3. Работа с файлами в Scilab

Рассмотрим функции Scilab для работы с файлами.

9.3.1. Функция открытия файла mopen

Обращение к функции открытия файла имеет вид:

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

file – строка, в которой хранится имя файла,

mode – режим работы с файлом:

- '**r**' открываемый текстовый файл доступен для чтения,
- '**rb**' открываемый двоичный файл доступен для чтения,
- '**w**' – создаваемый пустой текстовый файл предназначен только для записи информации;
- '**wb**' – создаваемый пустой двоичный файл предназначен только для записи

информации;

- '**a**' – открываемый текстовый файл будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- '**ab**' – открываемый двоичный файл будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- '**r+**' – открываемый текстовый файл используется в режиме чтения и записи;
- '**rb+**' – открываемый двоичный файл используется 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.

9.3.2. Функция записи в текстовый файла fprintf

Функция записи в текстовый файл **mfprintf** имеет вид

mfprintf(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:

%[ширина][.точность]тип.

Значения параметров строки преобразования приведены в таблице 9.1.

Таблица 9.1

Параметр	**Назначение**
Флаги	
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «-»
Пробел	Перед положительным числом выводится пробел, перед отрицательным – «-»
#	Выводится код системы счисления: 0 –

Параметр	**Назначение**
	перед восьмеричным числом, 0x (0X) пе-ред шестнадцатеричным числом.
Ширина	
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.
Точность	
ничего	Точность по умолчанию
n	Для типов e, E, f выводить n знаков после десятичной точки
Модификатор	
h	Для d, i, o, u, x, X короткое целое
l	Для d, i, o, u, x, X длинное целое
Тип	
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.ddde[+|-]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

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 **meof(f)** (**f** – идентификатор файла), которая возвращает единицу, если достигнут конец файла, и ноль в противном случае.

9.3.4 Функция закрытия файла mclose

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

mclose(f)

Здесь f – идентификатор закрываемого файла. С помощью функции mclose('all') можно

закрыть сразу все открытые файлы, кроме стандартных системных файлов.

Пример создания текстового файла приведен на листинге 9.15.

```
N=3;
M=4;
A=[2 4 6 7; 6 3 2 1; 11 12 34 10];
f=mopen('E:\abc.txt','w');
fprintf(f,'%d\t%d\n',N,M);
for i=1:N
for j=1:M
fprintf(f,'%g\t',A(i,j));
end
fprintf(f,'\n');
end
mclose(f);
```

Листинг 9.15.Создание текстового файла

Созданный текстовый файл можно .увидеть на рис. 9.18

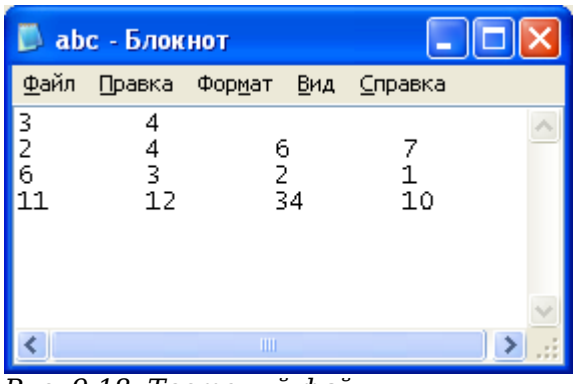


Рис. 9.18. Тестовый файл

Пример чтения данных из этого текстового файла приведен на листинге 9.16.

```
f=mopen('E:\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
mclose(f);
```

Листинг 9.16.Чтение из текстового файла

Результаты работы файла сценаия, представленного на листинг 9.16 представлены на листинге 9.17.

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

Листинг 9.17. Результат работы файла сценария

9.4. Пример программы в Scilab

В качестве примера 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.

Блок -схема решения задачи представлена на рис. 9.19, программа – на листинге 9.18. 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.

В Scilab несложно оформлять личные функции.

```
N=input('N=');
disp('Vvod massiva Y');
for i=1:N
y(i)=input('Y=');
end
disp(y); k=0;
for i=1:N
if y(i)>0
k=k+1;
x(k)=y(i);
end; end;
```

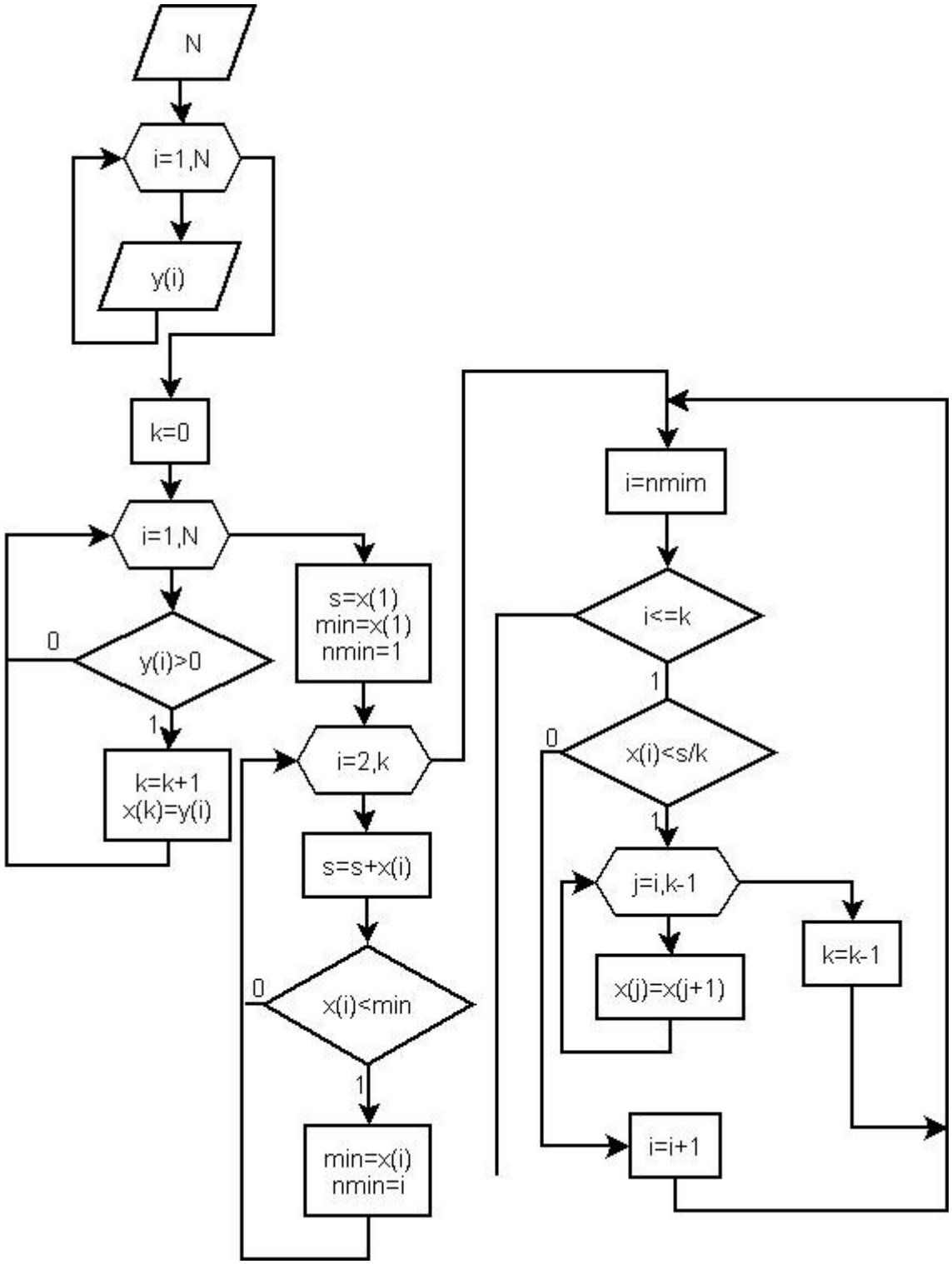


Рис. 9.19. Блок-схема решения задачи 9.3

```
disp(x);
s=x(1);
min1=x(1);
Nmin=1;
for i=2:k
```

```
s=s+x(i);
if x(i)<min1
min1=x(i);
Nmin=i;
end;
end;
i=Nmin;
while i<=k
if x(i)<s/k
for j=i:k-1
x(j)=x(j+1);
end;
//Удаление последнего элемента массива
x(k)=[];
k=k-1;
else
i=i+1;
end;
end;
disp(x);
```

Листинг 9.18. Решение задачи 9.1

9.5. Функции в Scilab

Структура функции в Scilab

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

операторы

endfunction

Здесь **x1, x2, ..., xm** - список входных параметров функции; **y1,y2, ..., yn** - список выходных параметров функции.

Если вызываемая функция находится не в текущем файле, то перед ее вызовом следует загрузить файл, в котором находится функция **exec('file',-1)**.

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, максимума и количества максимумов оформим функцию, заголовок ее будет иметь вид:

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

Блок-схема алгоритма приведена на рис.9.21, текст функции на листинге 9.19.

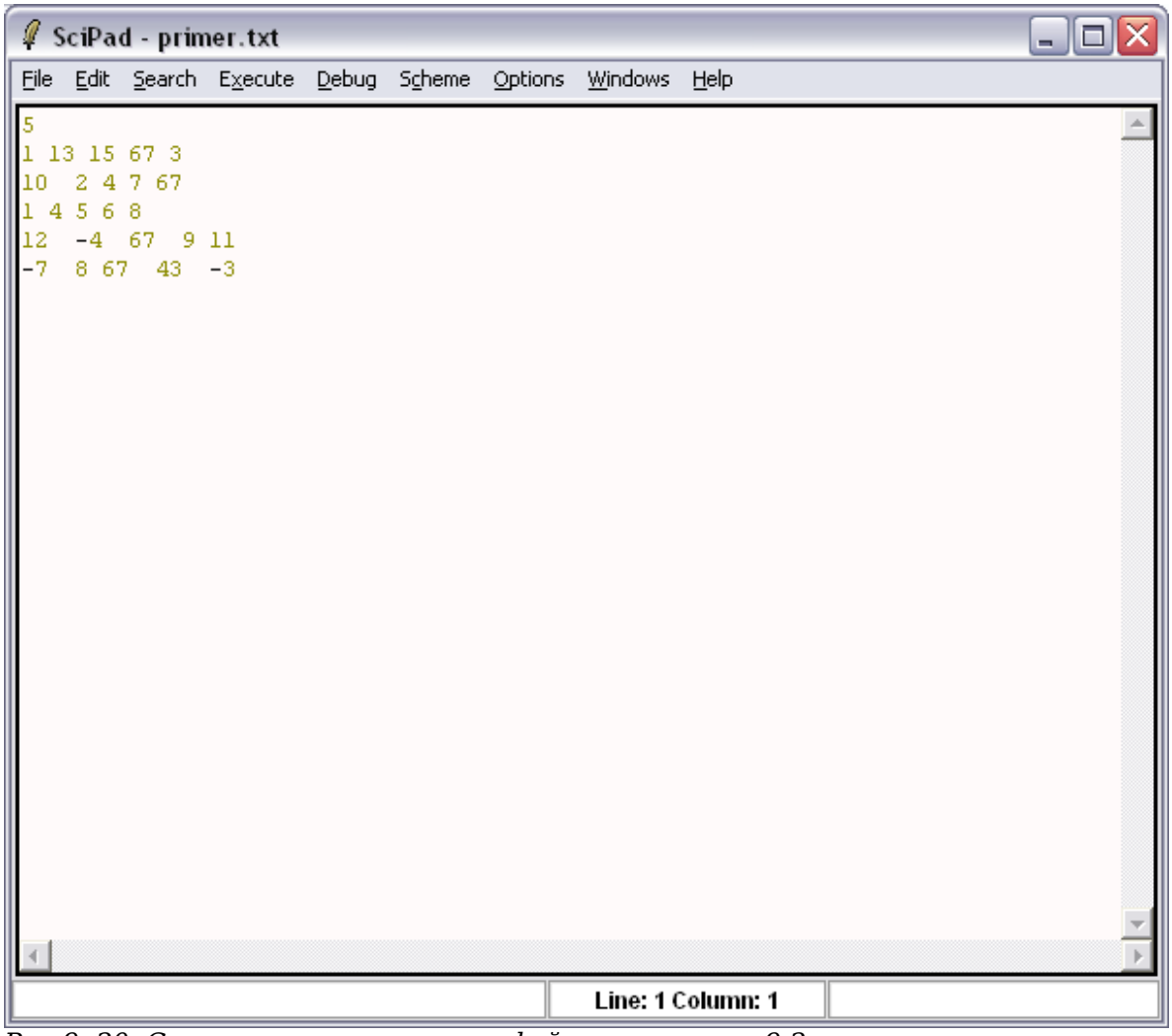


Рис.9. 20. Содержимое текстового файла для задачи 9.2.

```
function [summa, maximum, kolichestvo]=matrica_A(A,N)
summa=0;
for i=1:N
summa=summa+A(i,i)+A(i,N+1-i);
end
if (N-int(N/2)*2)==1
summa=summa-A(int(N/2)*2+1,int(N/2)*2+1);
end
maximum=A(1,1);kolichestvo=1;
for i=1:N
```

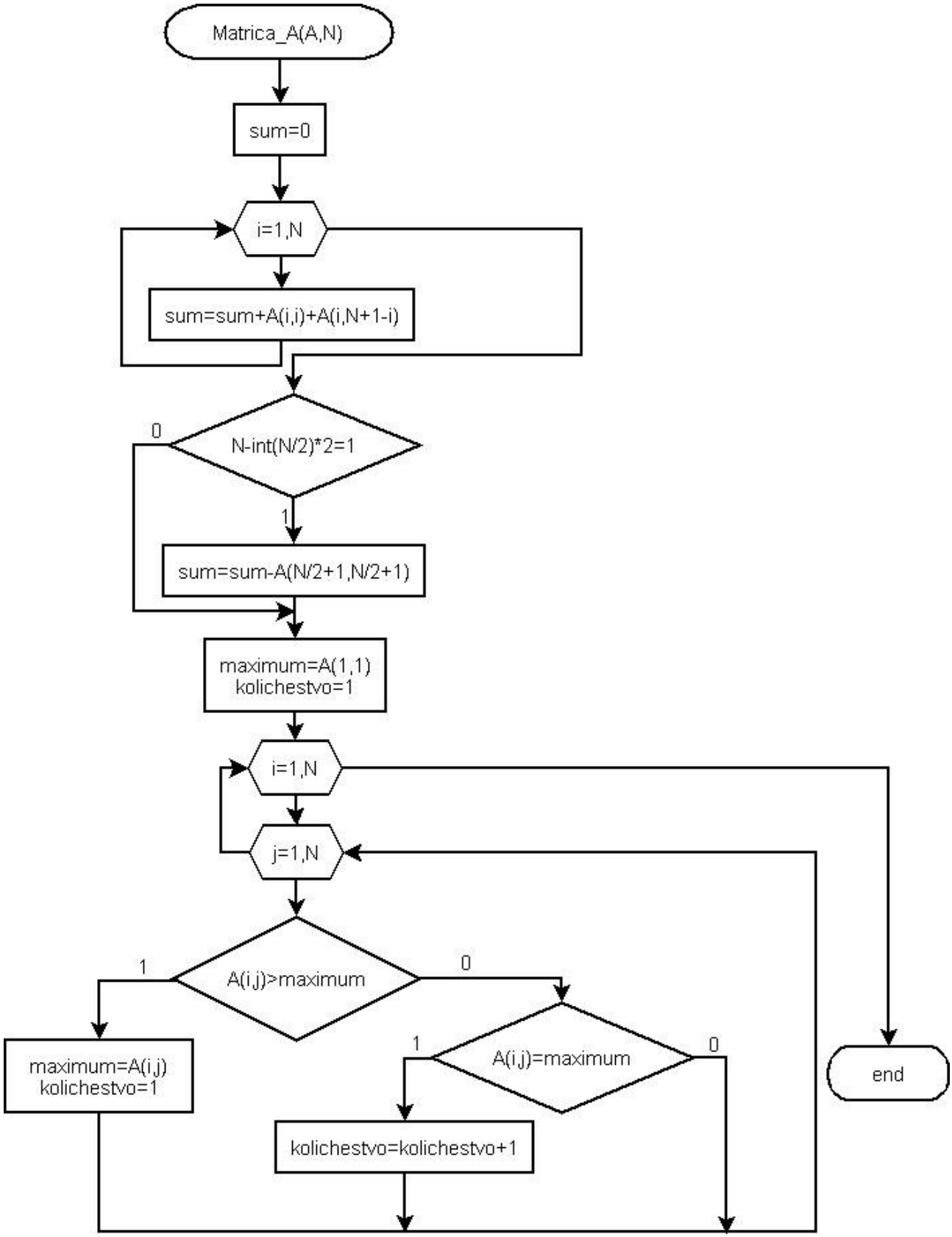


Рис. 9.21. Блок-схема функции matrica_A

```
for j=1:N
   if A(i,j)>maximum
```

```
maximum=A(i,j);
kolichestvo=1;
elseif A(i,j)==maximum
kolichestvo=kolichestvo+1;
end
end
end
endfunction
```

Листинг 9.19. Текст функции matrica_A

Текст функции, предназначенной для чтения матрицы из файла и вызова функции matrica_A приведен на листинге 9.20, результаты на листинге 9.21.

```
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 9.20. Вызов функции matrica_A

```
-->exec("matrica_2.sci");
-->s
 s  =
    21.
-->m
 m  =
    67.
-->k
 k  =
    4.
```

Листинг 9.21. Результаты работы программы