Понятие алгоритма, его свойства

- ***Алгоритм*** — конечная последовательность предписаний, определяющая процесс преобразования исходных и промежуточных данных в результаты решения задачи.

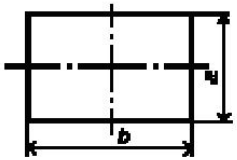
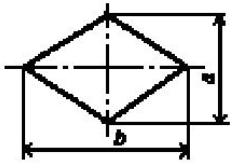
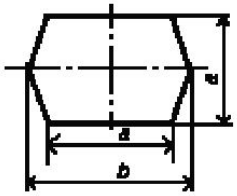
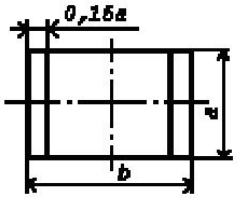
Свойства алгоритма

- Разрабатываемые алгоритмы должны обладать следующими свойствами:
- 1) *массовостью*, означающей, что если алгоритм разработан для решения определённой задачи, то он должен быть приемлемым для решения задач этого типа при всех допустимых значениях исходных данных;
- 2) *результативностью*, означающей получение результата после выполнения над исходными данными заданной последовательности действий или сообщения о невозможности решения;
- 3) *дискретностью*, обуславливающей дискретный (пошаговый) характер процесса получения результата, состоящий в последовательном выполнении конечного числа заданных алгоритмом действий;
- 4) *детерминированностью*, определяющей однозначность получаемого результата при одних и тех же исходных данных.

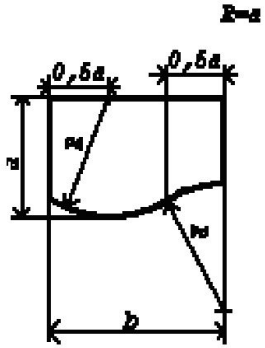
Способы описания алгоритмов

- словесный;
- графический;
- язык проектирования алгоритмов (псевдокоды);
- языки программирования.

- **Графический способ.** При этом способе записи алгоритм представляется в виде символов, блоков и связей между ними. Запись алгоритма должна выполняться в соответствии с государственными стандартами (ГОСТ 19.002–80 «Схемы алгоритмов и программ. Правила выполнения» и ГОСТ 19.003-80 «Схемы алгоритмов и программ. Обозначения условные и графические»).

Наименование	Обозначение и размеры в мм	Функция
1. Процесс		Выполнение операций или группы операций, в результате которых изменяется значение, форма представления или расположение данных
2. Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
3. Модификация		Выполнение операций, меняющих команды или группу команд, изменяющих программу
4. Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов или программ

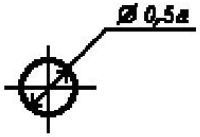
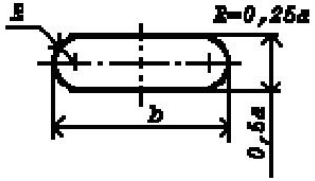
15. Документ



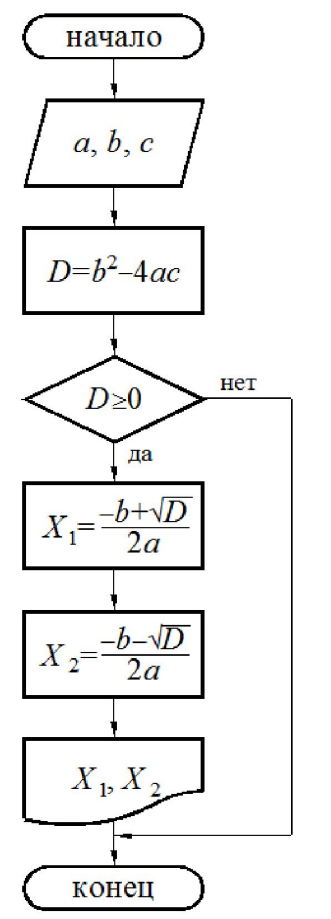
Ввод-вывод данных, носителем которых служит бумага

12. Ввод-вывод

или отображения результатов обработки (вывод)

28. Соединитель		Указание связи между прерванными линиями потока, связывающими символами
29. Пуск - останов		Начало, конец, прерывание процесса обработки данных или выполнения программы

Пример 4.1. Построить алгоритм нахождения действительных корней квадратного уравнения $ax^2 + bx + c = 0$ графическим способом (рис. 4.1).



Запись на языке программирования позволяет записывать алгоритм, который может быть реализован непосредственно на ЭВМ.

Каждый язык программирования имеет набор символов, правила написания языковых конструкций (синтаксис) и смысловые толкования этих конструкций. В настоящее время имеется большое число языков программирования, рассчитанных на разные классы решаемых задач. Запишем алгоритм решения квадратного уравнения на языке Бейсик:

```
10 INPUT A, B, C
20 D=B^2-4*A*C
30 IF D<0 GOTO 60
40 X1=(-B+SQR(D))/(2*A): X2=(-B-SQR(D))/(2*A)
50 PRINT X1, X2
60 END
```

C2

Дан целочисленный массив из 20 элементов. Элементы массива могут принимать целые значения от 0 до 10000 включительно. Опишите на естественном языке или на одном из языков программирования алгоритм, позволяющий найти и вывести максимальное значение среди трёхзначных элементов массива, не делящихся на 9. Если в исходном массиве нет элемента, значение которого является трёхзначным числом и при этом не кратно 9, то выведите сообщение «Не найдено».

Исходные данные объявлены так, как показано ниже на примерах для некоторых языков программирования и естественного языка. Запрещается использовать переменные, не описанные ниже, но разрешается не использовать некоторые из описанных переменных.

(допускаются иные формулировки ответа, не искажающие его смысла)

На языке Паскаль

```
max := 99;
for i := 1 to N do
    if (a[i]>=100) and (a[i]<=998) and (a[i] mod 9<>0) and
(a[i]>max) then
        max := a[i];
if max > 99 then writeln(max) else writeln('Не найдено');
```

На алгоритмическом языке

```
max := 99
нц для i от 1 до N
    если a[i]>=100 и a[i]<=998 и mod(a[i],9)<>0 и a[i]>max
    то
        max := a[i]
    все
кц
если max > 99
то
    вывод max
иначе
    вывод "Не найдено"
все
```

На языке Бейсик

```
MAX = 99
FOR I = 1 TO N
    IF A(I)>=100 AND A(I)<=999 AND A(I) MOD 9<>0 AND A(I)>MAX THEN
        MAX = A(I)
    END IF
NEXT I
IF MAX > 99 THEN
    PRINT MAX
ELSE
    PRINT "Не найдено"
END IF
```

На языке Си

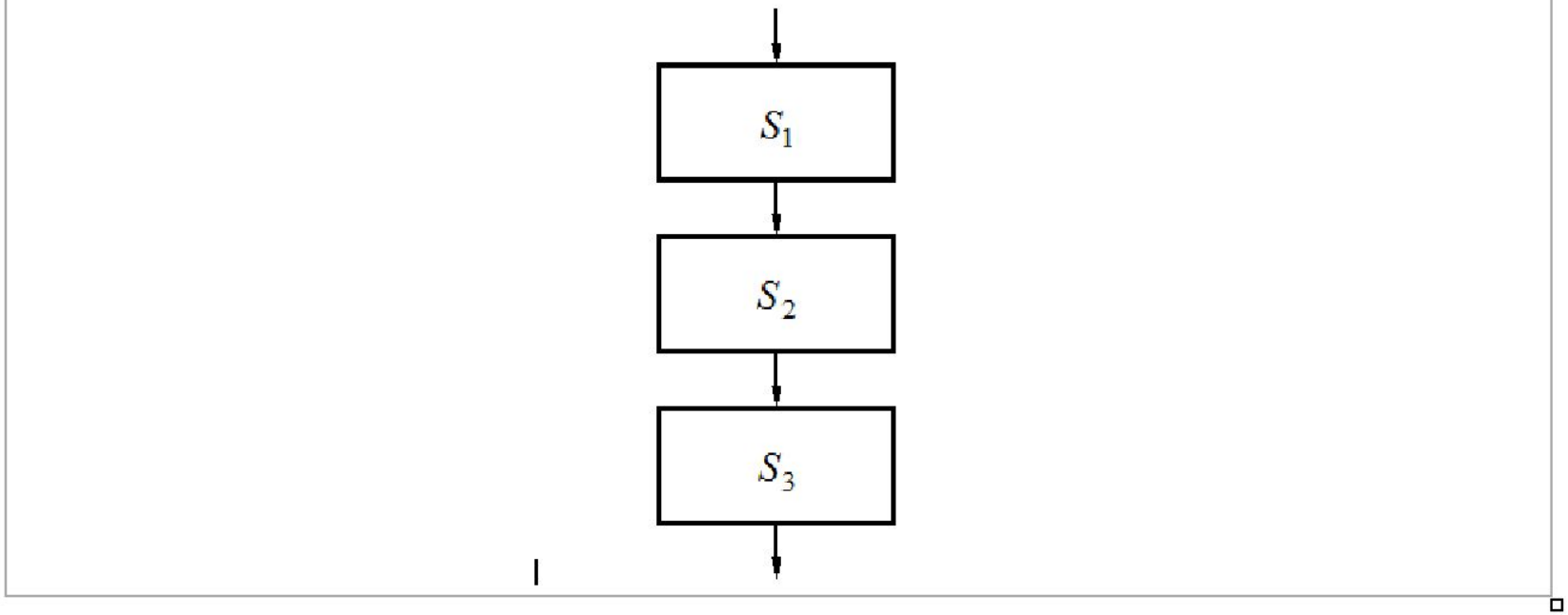
```
max = 99;
for (i = 0; i<N; i++)
    if (a[i]>99 && a[i]<999 && a[i]%9!=0 && a[i]>max)
        max = a[i];
if (max>99)
    printf("%d", max);
else
    printf("Не найдено");
```

Структурный подход к разработке алгоритмов, основные управляющие структуры

- Теорема о структурировании: логическая структура алгоритма может разрабатываться с использованием ограниченного числа элементарных управляющих структур: следование, разветвление, повторение (цикл).

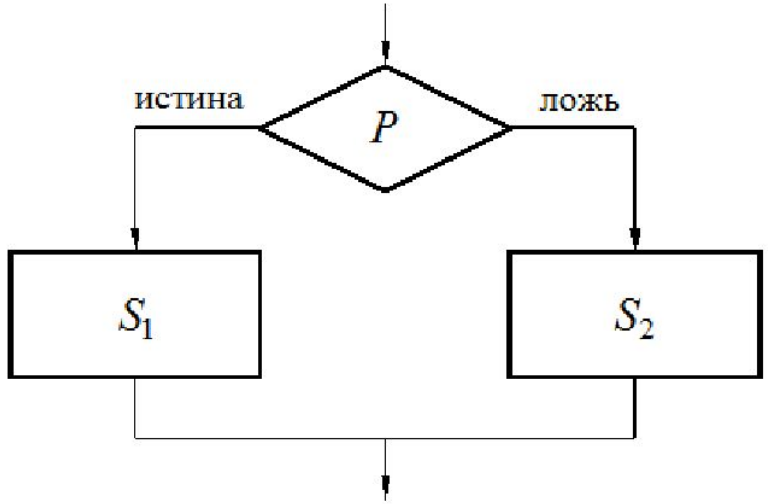
Следование

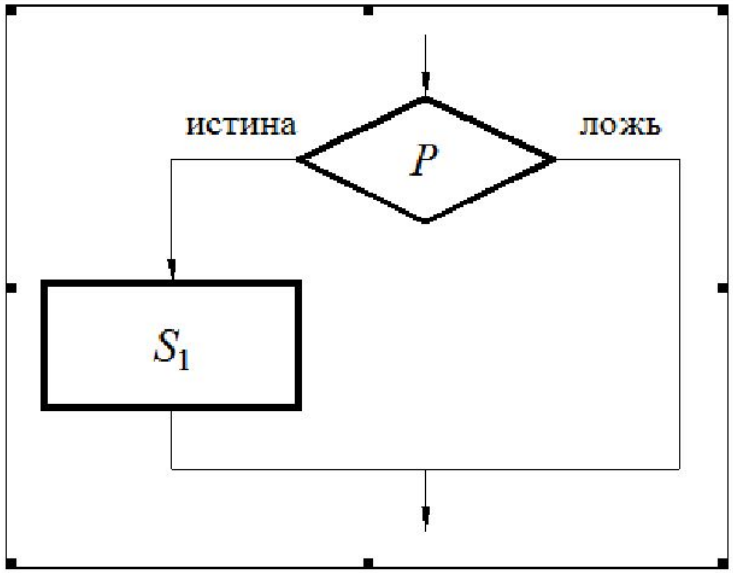
Управляющая структура следования обеспечивает естественную последовательность выполнения действий. Каждый прямоугольник содержит одно или несколько последовательно выполненных действий S_1—S_2 (рис. 4.2).



Разветвление

Управляющая структура разветвления (рис. 4.3) обеспечивает выбор действия S_1 или S_2 в зависимости от некоторого условия P. Если условие принимает значение «истинно», то выполняется действие S_1, в противном случае — S_2.





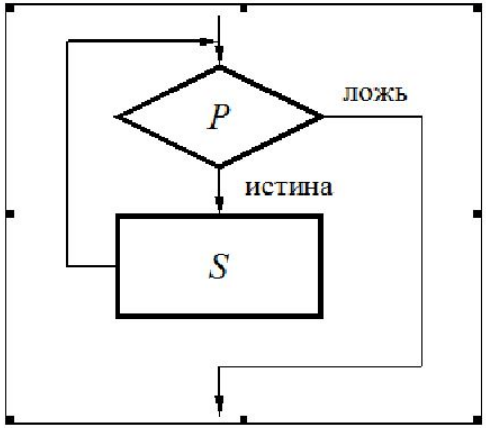
Цикл

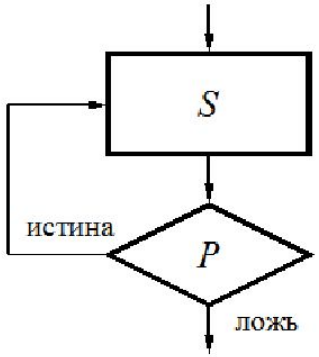
Управляющая структура повторения (цикл) предусматривает повторное выполнение действия *S* до тех пор, пока некоторое условие *P* имеет значение «истинно». Как только значение условия становится «ложно», прекращается выполнение действия *S* и управление передается следующей структуре.

Выделяют 2 вида циклов (управляющих структур повторения):

1) *цикл «ПОКА»*, — условие *P* проверяется до выполнения тела цикла *S*, соответственно, если на первом шаге условие *P* имеет значение «ложно», то происходит выход из цикла, действие *S* вообще не выполняется (рис. 4.5);

2) *цикл «ДО»*, — действие *S* выполнится хотя бы один раз даже при начальном значении условия *P* «ложно» (рис. 4.6).





Базовые управляющие структуры

- В зависимости от того, какие базовые управляющие структуры включаются в алгоритм, различают следующие виды структур алгоритмов:
- 1) линейные;
- 2) разветвляющиеся;
- 3) циклические

- **Линейным** называют алгоритм, при котором действия выполняются последовательно друг за другом, в естественном порядке.
- Таким образом, линейный алгоритм предусматривает использование только одной структуры следования**.**
- **Разветвляющимся** называют алгоритм, при котором происходит выбор одного из нескольких заранее предусмотренных направлений в зависимости от выполнения некоторого условия.
- Разветвляющийся алгоритм предусматривает использование структур следования и разветвления

- **Циклическим** называют алгоритм, в котором предусмотрено многократное выполнение одной и той же последовательности однотипных действий. Повторяющиеся последовательности действий называют *циклами*.
- Циклический алгоритм предусматривает обязательное использование структуры повторения.
- Для организации цикла необходимо предусмотреть:
- задание начального параметра цикла – переменной, которая будет изменяться при повторениях цикла;
- изменение значений этой переменной перед каждым новым повторением цикла;
- проверку условия окончания повторений по значению параметра и переход к началу цикла, если повторения не закончены.
- Изобразим структуру цикла в виде схемы (рис. 4.9).

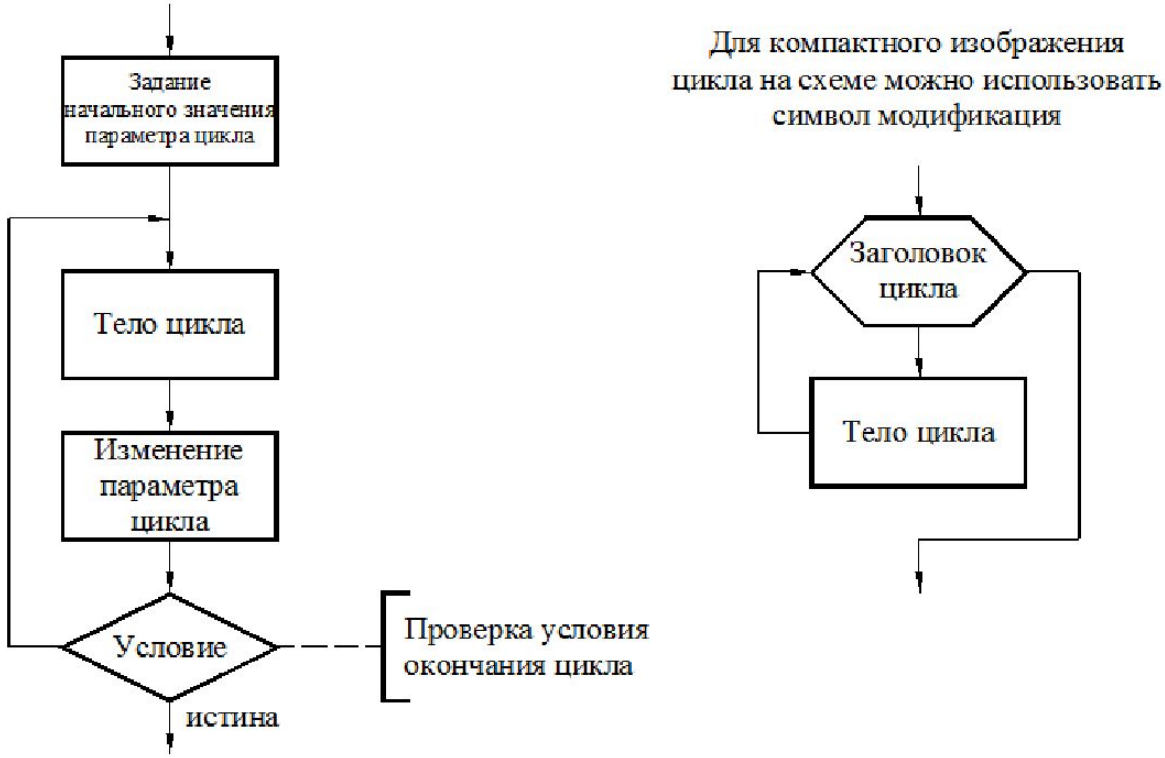


Рис. 4.9 Структура цикла

ORIGIN := 1

n := 3

Ввод матрицы A и вектора X при помощи цикла

$$\begin{pmatrix} A \\ X \end{pmatrix} := \left| \begin{array}{l} \text{for } i \in 1..n \\ \quad \left| \begin{array}{l} X_i \leftarrow i \cdot n \\ \text{for } j \in 1..n \\ \quad A_{i,j} \leftarrow i + j \end{array} \right. \\ \begin{pmatrix} A \\ X \end{pmatrix} \end{array} \right.$$

$$A = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{pmatrix} \qquad X = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

Рис. 4.15 Пример ввода массивов при помощи операторов цикла

Вывод значений из программы

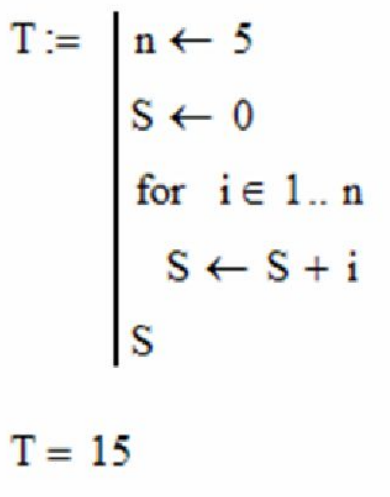


Рис. 4.16 Пример вывода переменной на печать

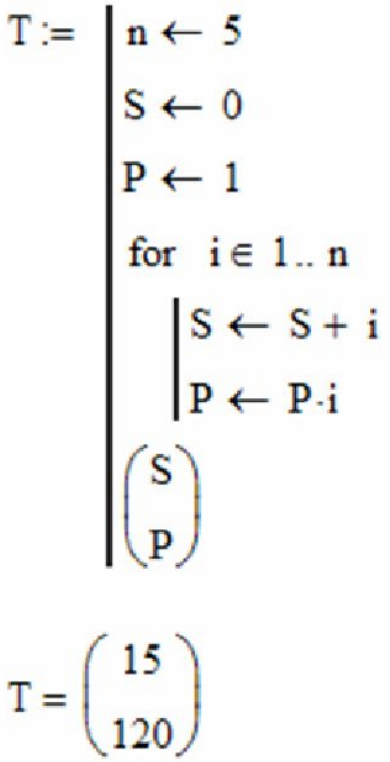


Рис. 4.17 Пример вывода нескольких переменных на печать

- ***Матрицей*** называется массив элементов (в частности, чисел), расположенных в виде прямоугольной таблицы из строк и столбцов.
- Все элементы матрицы обозначаются одним именем. Так, на рис. 4.18 изображена матрица размерности . Местоположение каждого элемента матрицы характеризуется двумя индексами: номером строки и номером столбца , на пересечении которых находится элемент. Так как элементы матрицы обозначены одним именем и различаются только индексами, обработку их целесообразно производить циклическим алгоритмом, а в качестве параметра цикла использовать индексы элементов

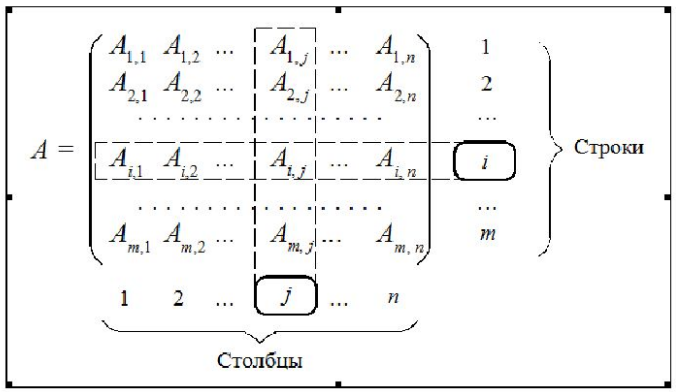


Рис. 4.18

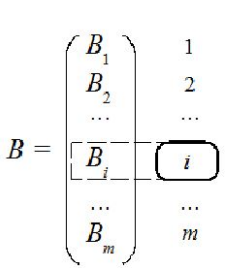


Рис. 4.19

Разновидностями прямоугольной матрицы $(m \times n)$ являются:

- матрица-столбец или вектор $(m \times 1)$ (рис. 4.19);
- матрица-строка $(1 \times n)$ (рис. 4.21);
- квадратная матрица $(n \times n)$ (рис. 4.20).

$$B = \begin{pmatrix} B_1 & B_2 & \dots & B_n \end{pmatrix}$$

Элементы матрицы-столбца (рис. 4.19) можно рассматривать в качестве координат конца вектора $\vec{B} = \{B_1, B_2, \ldots, B_m\}^T$ в линейном n-мерном пространстве.

Линия, на которой располагаются элементы $A_{1,1}$, $A_{2,2}$, …, $A_{n,n}$ квадратной матрицы A называется *главной диагональю* (рис. 4.20). Рекуррентная формула элемента главной диагонали:

$$A_{i,i}, (i = 1, \ldots, n). \qquad (4.2)$$

Линия, на которой располагаются элементы $A_{1,n}$, $A_{2,(n-1)}$, …, $A_{n,1}$ квадратной матрицы A, называется *побочной диагональю* (рис. 4.20). Рекуррентная формула элемента побочной диагонали:

$$A_{i,(n-i+1)} \ (i = 1, \ldots, n). \qquad (4.3)$$

Алгоритм ввода элементов матриц и векторов

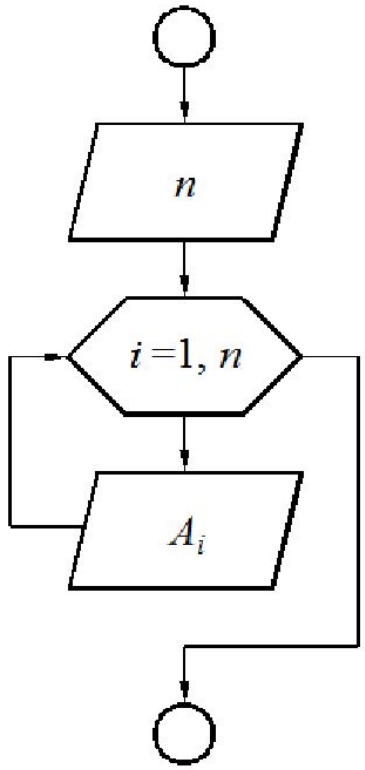


Рис. 4.22 Блок-схема ввода вектора

ORIGIN := 1

$$A := \left| \begin{array}{l} n \leftarrow 3 \\ \text{for } i \in 1..n \\ \quad A_i \leftarrow i \cdot n \\ A \end{array} \right.$$

$$A = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

Рис. 4.23 Реализация алгоритма ввода вектора

- Если один цикл помещается внутри другого, то такие циклы называются *вложенными*. Цикл, содержащий другие циклы, называется *внешним*, а входящие в него циклы — *внутренними*.
- Работа вложенного цикла заключается в следующем: сначала задается первое значение параметра внешнего цикла, далее управление передается внутреннему циклу, и параметр внутреннего цикла принимает по очереди все значения. Когда выполнение внутреннего цикла окончено, то задается второе значение параметра внешнего цикла, и вновь выполняется внутренний цикл. Процесс повторяется до тех пор, пока параметр внешнего цикла не примет все значения. Например, если количество повторений внешнего цикла , а внутреннего , то общее число повторений будет равно .

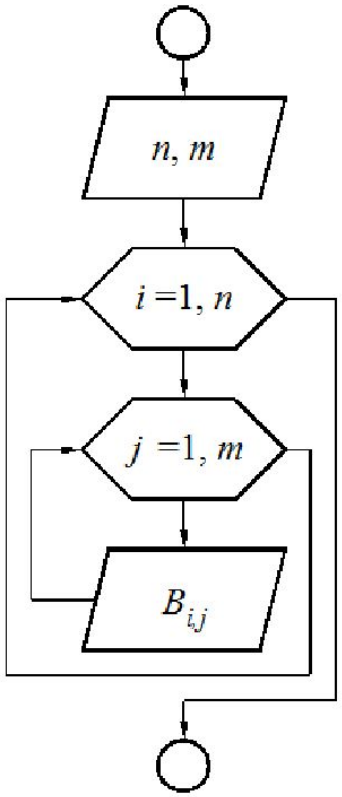


Рис. 4.24 Блок-схема ввода матрицы

ORIGIN := 1

B :=
- n ← 3
- m ← 4
- for i ∈ 1..n
  - for j ∈ 1..m
    - $B_{i,j}$ ← i·j
- B

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

Рис. 4.25 Реализация алгоритма ввода матрицы

Алгоритм вычисления суммы элементов вектора

Пример 4.7. Составить схему алгоритма вычисления суммы значений заданной аналитически функции $y = f(x)$ в точках, заданных вектором $\vec{X} = \{X_1, X_2, \ldots, X_n\}^T$.

Математическая формулировка задачи будет следующей:

$$S = \sum_{i=1}^{n} f(X_i). \qquad (4.5)$$

Решение задачи организуется следующим образом: будем вычислять значения функции $Y_i = f(X_i)$ в каждой точке X_i и прибавлять их к сумме предыдущих значений этой функции, т. е. будем последовательно получать все промежуточные суммы S_i:

$$S_1 = 0 + f(X_1) = f(X_1);$$

$$S_2 = S_1 + f(X_2) = f(X_1) + f(X_2);$$

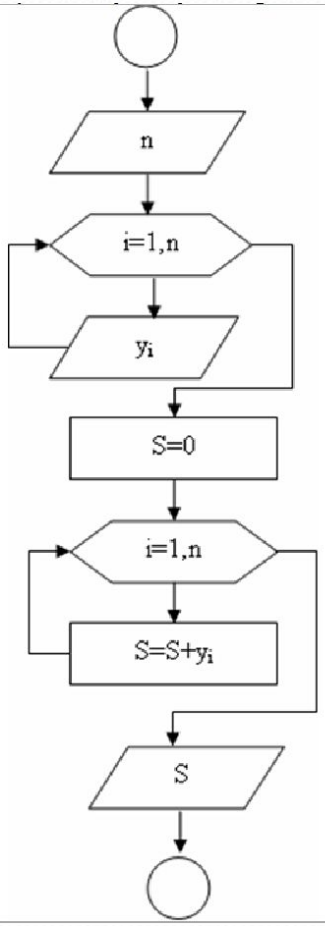
$$S_3 = S_2 + f(X_3) = f(X_1) + f(X_2) + f(X_3); \qquad (4.6)$$

……

$$S_n = S_{n-1} + f(x_n) = f(X_1) + f(X_2) + \ldots + f(X_n).$$

- Рекуррентная формула, по которой осуществляется накопление суммы, будет иметь вид

$$S = S + Y_i.$$



ORIGIN := 1

n := 3

$$y := \left| \begin{array}{l} \text{for } i \in 1..n \\ \quad y_i \leftarrow i \cdot n \\ y \end{array} \right. \qquad y = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

$$S := \left| \begin{array}{l} S \leftarrow 0 \\ \text{for } i \in 1..n \\ \quad S \leftarrow S + y_i \\ S \end{array} \right. \qquad S = 18$$

Алгоритм вычисления произведения элементов вектора

Пример 4.8. Составить блок-схему алгоритма вычисления произведения значений элементов вектора $Y = \{Y_1, Y_2, \ldots, Y_n\}^T$.

Математическая формулировка задачи:

$$P = \prod_{i=1}^{n} Y_i. \qquad (4.8)$$

Нахождение произведения будет производиться аналогично накоплению суммы, с той лишь разницей, что для вычисления произведения будет использоваться рекуррентная формула

$$P = P \cdot Y_i. \qquad (4.9)$$

Начальное значение переменной P, в которой будет накапливаться произведение, должно быть равным единице (см. рис. 4.30, 4.31).

Произведение элементов вектора

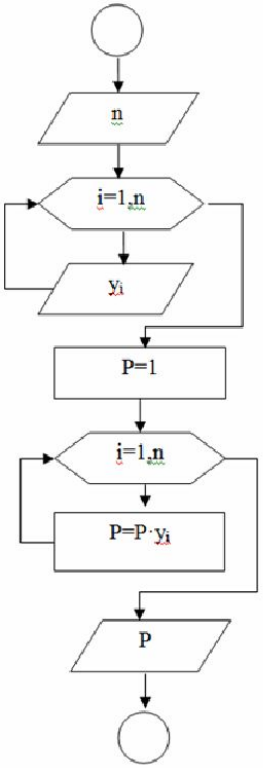


Рис. 4.30 Блок-схема алгоритма вычисления произведения элементов вектора

ORIGIN := 1

n := 3

$$y := \left| \begin{array}{l} \text{for } i \in 1..n \\ \quad y_i \leftarrow i \cdot n \\ y \end{array} \right. \qquad y = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

$$P := \left| \begin{array}{l} P \leftarrow 1 \\ \text{for } i \in 1..n \\ \quad P \leftarrow P \cdot y_i \\ P \end{array} \right. \qquad P = 162$$

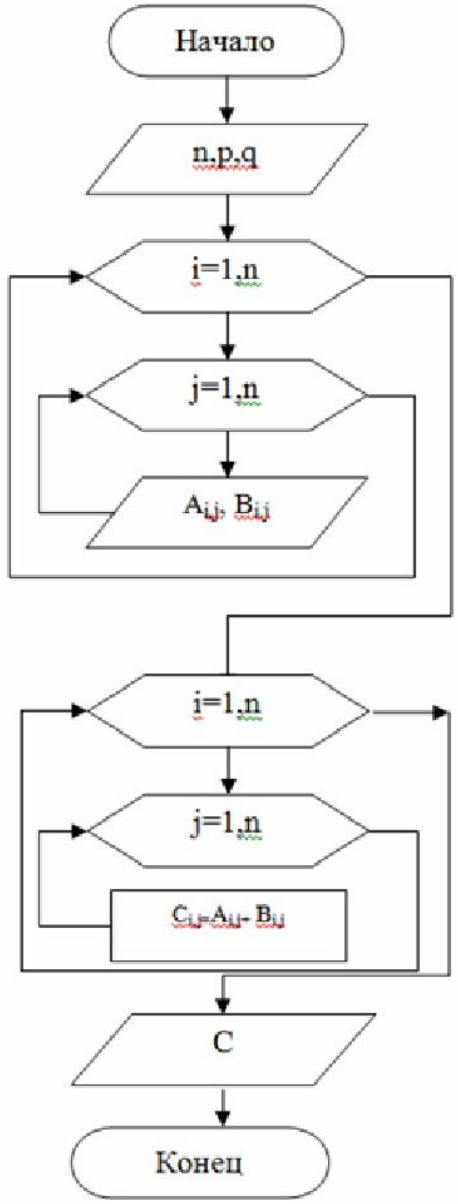
Рис. 4.31 Реализация алгоритма вычисления произведения элементов вектора

Сумма двух матриц

Пример 4.9. Составить блок-схему алгоритма нахождения суммы двух матриц $C = A + B$. Задаем известные матрицы A и B. Просуммировать можно

матрицы только одинаковой размерности. Поэтому сумма двух матриц $A_{n\times m}$ и $B_{n\times m}$ даст новую матрицу $C_{n\times m}$ той же размерности, каждый элемент которой находится по следующей рекуррентной формуле:

$$C_{i,j} = A_{i,j} + B_{i,j}. \qquad (4.10)$$



ORIGIN := 1

n := 3

$$\begin{pmatrix} A \\ B \end{pmatrix} := \left| \begin{array}{l} \text{for } i \in 1..n \\ \quad \text{for } j \in 1..n \\ \quad\quad A_{i,j} \leftarrow i \cdot n \\ \quad\quad B_{i,j} \leftarrow i + j \\ \begin{pmatrix} A \\ B \end{pmatrix} \end{array} \right.$$

$$A = \begin{pmatrix} 3 & 3 & 3 \\ 6 & 6 & 6 \\ 9 & 9 & 9 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{pmatrix}$$

$$C := \left| \begin{array}{l} \text{for } i \in 1..n \\ \quad \text{for } j \in 1..n \\ \quad\quad C_{i,j} \leftarrow A_{i,j} + B_{i,j} \\ C \end{array} \right. \qquad C = \begin{pmatrix} 5 & 6 & 7 \\ 9 & 10 & 11 \\ 13 & 14 & 15 \end{pmatrix}$$