# Содержание

# Введение

Данное методическое пособие предназначено для студентов направления 230100.62 "Информатика и вычислительная техника" А так же студентов других направлений при изучении дисциплин связанных с разработкой программ.

Целью пособия является ознакомление обучающихся с основами алгоритмизации и правилами оформления схем алгоритмов.

Приведем некоторые определения.

Алгоритм - описание последовательности действий, строгое исполнение которых приводит к решению поставленной задачи за конечное число шагов. Для любого алгоритма характерны следующие свойства: дискретность, детерминированность. конечность, массовость и результативность.

Существуют различные формы представления алгоритмов: словесный, формульно-словесный, графический, алгоритмический язык. Словесный способ представления несложен, но имеет ряд недостатков. Главный из которых состоит в том, что при таком способе допускается некоторая произвольность изложения и не существует четких стандартов описания. Сложные задачи с анализом условий, с повторяющимися действиями и возвратами к предыдущим пунктам трудно представляются в словесном и словесно-формульном виде. Наиболее наглядным является графический способ записи алгоритмов, одной из форм которого является схема алгоритма.

Схемой алгоритма называется графическое изображение логической структуры алгоритма, в котором каждый этап процесса обработки информации представляется в виде геометрических символов, имеющих определенную конфигурацию в зависимости от характера выполняемых операций.

Несмотря на часто кажущуюся студентам ненужность аппарата схем, важно помнить, что схема алгоритма - это, прежде всего, четко оформленная идея. Используемый в дискуссиях аргумент о нецелесообразности разработки схем алгоритмов в деятельности, не связанной с учебным процессом, не лишен смысла. Зачастую действительно быстрее написать и отладить программа "с ходу", чем формально описывать алгоритм работы. Однако цель учебного процесса - научить студента ясному алгоритмическому мышлению. Именно эту задачу великолепно решают схемы алгоритма. Кроме того, существенным аргументом в пользу использования схем является то, что, разрабатываемый алгоритм не привязан к синтаксису определенного языка.

Помните, что даже специалисты большую часть рабочего времени тратят на обдумывание способов решения поставленных перед ними задач. Только после грамотного анализа и оценки возможных вариантов реализации целесообразно приступать к написанию программного кода. Поэтому необходимо стремиться к достижению состояния предельно ясного осознания метода осуществления задуманного. Возможно, после приобретения способности разработки алгоритмов только посредством устного рассуждения и имеет смысл отказаться от составления блок-схем как повседневной практики. Однако, на этапе освоения азов программирования необходимо прибегать к данному аппарату хотя бы в целях дисциплинирования ума.

Следует обратить внимание на тот факт, что излагаемый материал подается с содержательной точки зрения и содержит ряд незначительных терминологических допущений. В частности, термин "алгоритм" нередко подменяется термином "программа".

Пособие рекомендуется к ознакомлению в начале процесса изучения дисциплины "Программирование".

# 1 Изображение схемы алгоритма без ветвлений

В данном разделе представлены общие сведения о составлении схем алгоритмов и программ: требования к оформлению базовых символов, их соединений и содержимого, а также обязательным элементам блок-схемы.

## 1.1 Символы начала и конца программы

Схема любого алгоритма всегда начинается терминальным символом "Начало", а заканчивается терминальным символом "Конец". Примеры их изображения представлены на рис.1.1.

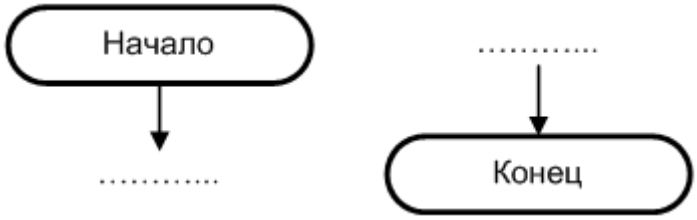


Рис.1.1. Пример изображения терминальных символов

Данные элементы являются своеобразными точками входа и выхода из блок-схемы, поэтому символ "Начало" имеет только выходную линию, а символ "Конец" - входную.

## 1.2 Операторный символ

Операторный символ (или символ процесса) является базовым элементом схемы, который предназначен для отображения действий, совершаемых над данными. Текст символа может описывать как одну, так и несколько операций. Во втором случае допускается увеличение высоты символа. Пример изображения элемента представлен на рис.1.2.

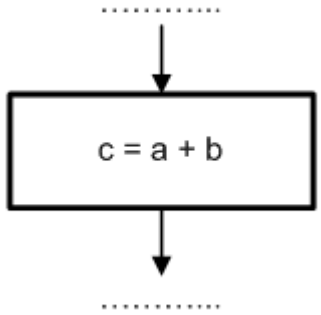


Рис.1.2. Пример использования операторного символа

На практике операторный символ чаще всего используется для описания процесса вычислений.

## 1.3 Символ ввода-вывода данных

Для обозначения операции ввода-вывода данных используется символ данных, пример изображения которого приведен на рис.1.3.

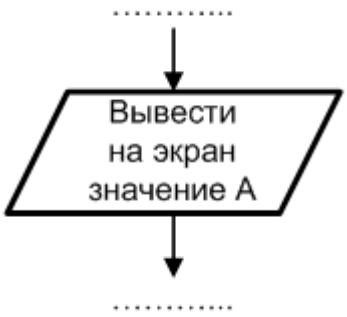


Рис.1.3. Пример использования символа данных

Данный элемент применяется для указания выполнения операции как ввода данных пользователем, так и чтения данных из внешнего источника. Кроме того, символ позволяет отразить процесс вывода информации на носитель любой природы (экран, ВЗУ и т.д.). При этом природа обрабатываемых данных (числовая, текстовая, графическая и т.д. информация) не имеет значения.

## 1.4 Комментарии

Зачастую размер символа схемы  не позволяет отразить некоторые нюансы алгоритмы. В связи с этим особо важные с содержательной точки зрения элементы могут быть сопровождены комментариями. Дополнительная информация изображается на схеме в соответствии с представленным на рис.1.4 способом.

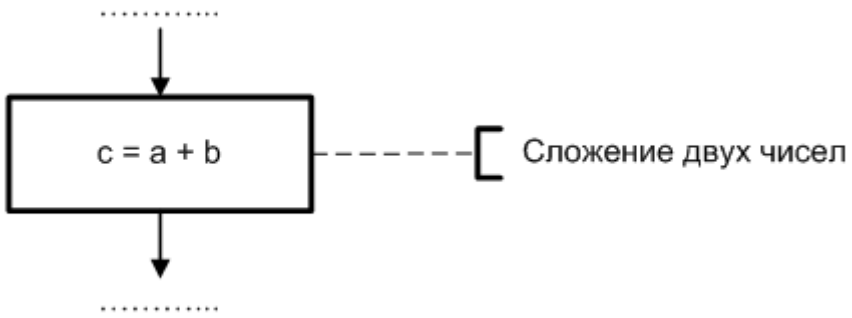


Рис.1.4.  Пример использования комментария

Следует помнить, что текст комментария приводится на русском языке, а сам символ располагается в наиболее удобном месте схемы, не перекрывая остальные элементы схемы. Если это не нарушает общего восприятия схемы, то рекомендуется помещать пояснения в непосредственной близости от комментируемых символов.

Кроме того, допускается заключать группу символов, к которой относится комментарий, в пунктирную рамку. Пример такого изображения представлен на рис.1.5.

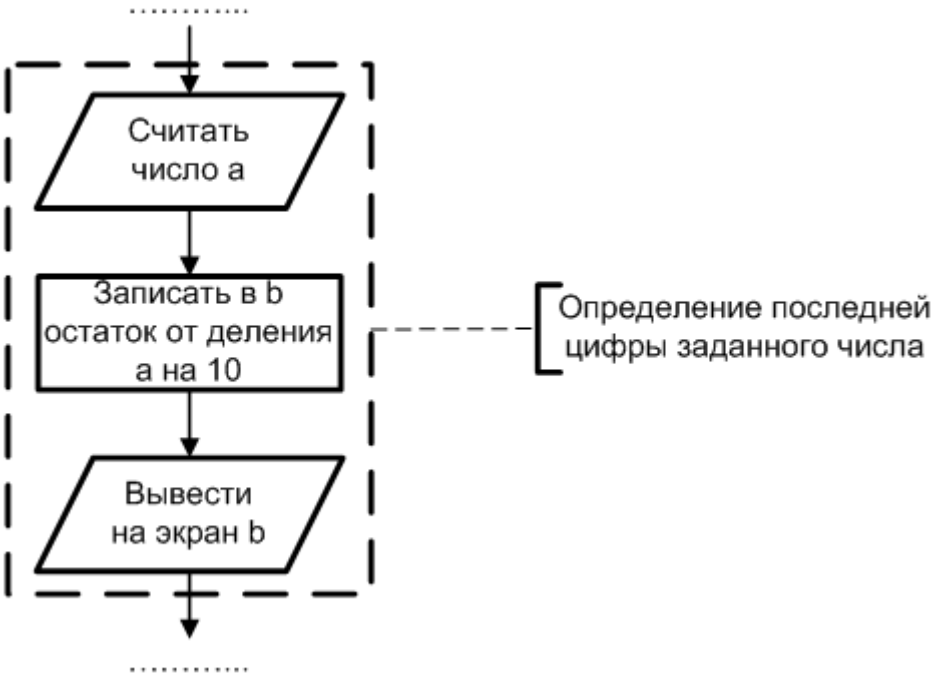


Рис.1.5 . Пример использования комментария группы символов

## 1.5 Общие рекомендации по оформлению схем алгоритмов

При оформлении схем алгоритмов рекомендуется придерживаться следующих правил.

1. Несмотря на то, что стандарт не накладывает ограничений на содержимое символов, рекомендуется описывать команды на языке, близком к естественному. Так, действие по выводу значения переменной на экран может быть записано как "writeln(X)" или "Вывести X". Второй вариант является более предпочтительным. Данная рекомендация связана с тем, что описание команд на языке, близком к конкретному языку программирования (ЯП) чревато путаницей: языки программирования имеют разную популярность и распространенность, зависящую от огромного числа факторов. Например, команда <> для подавляющего большинства неспециалистов в этом ЯП может оказаться незнакомой. Схема алгоритма - по своему назначению прежде всего универсальная структура, поэтому следует избегать ее связывания с конкретным ЯП.

2. Соединение элементов схемы выполняется посредством линий. Стандартным считается направление слева направо и сверху вниз. Для внесения ясности рекомендуется использовать стрелки, размещаемые на окончании линий.

3. Для всех символов за исключением условного входной считается верхняя или левая сторона, а выходной - нижняя или правая (при этом приоритет следует отдавать верхней и нижней сторонам).

4. По возможности следует избегать пересечения соединительных линий. В случае, когда разрыв линии позволяет оптимизировать графическое представление алгоритма, либо алгоритм является многостраничным, допускается использовать символ соединитель, изображаемый в виде круга с меткой . Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва - внутренним соединителем. Пример данного действия представлен на рис.1.6.

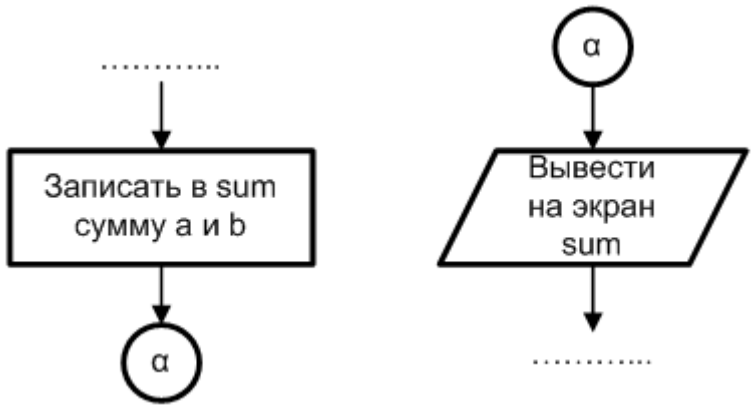


Рис.1.6 . Пример использования соединителя

5. При работе с соединителями следует также принимать во внимание перечисленные требования:

— необходимо обеспечивать уникальность используемой метки в пределах схемы (желательно - в пределах всего набора документации);

— допускается использовать несколько внешних соединителей с одной меткой, однако метки всех внутренних соединителей должны быть попарно различны;

— соединители рекомендуются к применению при переносе части схемы на другую страницу (при этом в комментарии следует указывать номер следующей страницы).

6. При печати допускается изображение схем как в книжной, так и в альбомной ориентациях.

7. Если размер символа не позволяет включить в него минимум текста для описания его смысловой нагрузки, то допускается изложение части содержимого (или всего текста) в символе комментария.

8. Если это необходимо, допускается присваивание символу уникального идентификатора, который размещается слева и сверху от соответствующего блока. Данный идентификатор может быть использован в сопроводительном тексте для явной ссылки на символ. Пример изображения идентификатора представлен на рис.1.7.

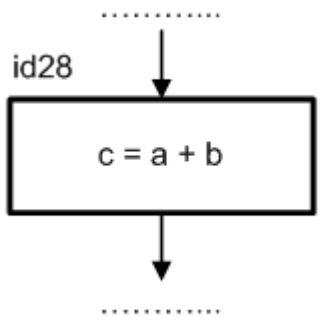


Рис.1.7. Пример использования идентификатора

Типовые ошибки, как правило, связаны с нарушением следующих правил:

— ширина всех блоков схемы должна быть одинаковой;

— при соединении блоков следует использовать только вертикальные и горизонтальные линии потоков;

— горизонтальные потоки, имеющие направление справа налево, и вертикальные потоки, имеющие направление снизу вверх, должны быть обязательно помечены стрелками;
— линии потоков должны быть параллельны линиям внешней рамки или границам листа;
— текст символов и комментариев должен записываться слева направо и сверху вниз независимо от направления потоков данных или управления;
— как правило, каждый символ имеет один вход и один выход, исключение составляют:
    - терминальные символы (у операции "начало" нет входа, у операции "конец" нет выхода);
    - символы решения (один вход и несколько выходов).

## 1.6 Пример схемы алгоритма линейной программы

По структуре алгоритмы разделяют на линейные, разветвляющиеся и циклические.

Линейными называют алгоритмы, в которых операции выполняются последовательно одна за другой, в естественном и единственном порядке следования. Как правило, алгоритмы линейной структуры состоят из трех частей: ввод исходных данных, вычисления результатов по формулам, вывод значений результатов. Это самые простые алгоритмы, которые являются составной частью любого алгоритмического процесса.

Пусть требуется реализовать программу для подсчета суммы, разности и произведения двух заданных чисел.

Схема алгоритма решения предлагаемой задачи изображена на рис.1.8.

Тогда программа, написанная в соответствии с представленной на рис.схемой, будет выглядеть следующим образом.

На языке программирования Pascal:

```
var a, b, sum, sub, mul: integer;
begin
read(a, b);
  sum := a + b;
  write(sum, ' ');
  sub := a - b;
  write(sub, ' ');
  mul := a * b;
  write(mul);
end.
```

На языке программирования C:

```
#include <stdio.h>
int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    int sum = a + b;
    printf("Sum = %d\n", sum);
    int sub = a - b;
    printf("Sub = %d\n", sub);
    int mul = a * b;
    printf("Mul = %d\n", mul);
    return 0;}
```

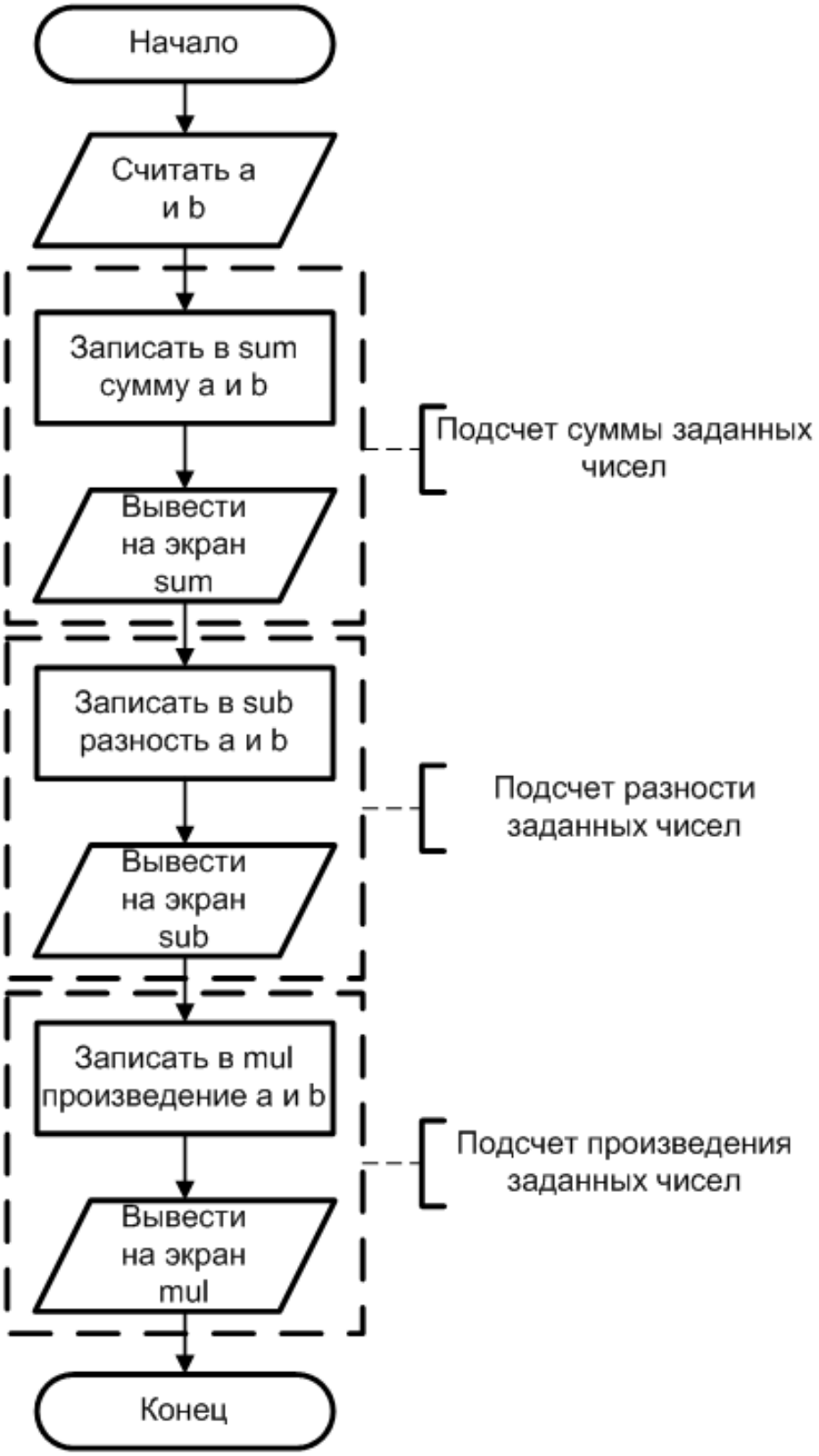


Рис.1.8. Схема алгоритма для нахождения суммы, разности и произведения двух заданных чисел

## 1.7 Задачи для самостоятельной подготовки

Изобразите схемы алгоритмов для решения предлагаемых задач.

1. Определите последнюю цифру заданного числа.
2. Определите сколько десятков в заданном числе.
3. Найдите сумму цифр трехзначного числа.
4. Найдите следующее четное число за заданным.
5. Найдите абсолютное значение заданного числа.
6. Определите площадь прямоугольника по длинам его сторон.
7. Определите площадь треугольника по длинам его сторон.
8. Определите объем шара с радиусом R.
9. Найдите середину отрезка, заданного координатами своих концов.
10. Найдите координаты точки пересечения двух прямых.

# 2 Изображение схемы алгоритма с ветвлениями

В данном разделе рассматриваются вопросы, связанные с оформлением ветвящихся процессов и циклических конструкций.

## 2.1 Условный оператор

Подавляющее большинство алгоритмов имеет структуру, отличную от линейной. На практике часто возникает необходимость проведения анализа исходных данных или промежуточных результатов вычислений с целью определения дальнейшего порядка выполнения вычислительного процесса. Алгоритмы, в которых в зависимости от выполнения некоторого логического условия происходит разветвление вычислений по одному из нескольких возможных направлений, называют разветвляющимися. Подобные алгоритмы предусматривают выбор одного из альтернативных путей продолжения вычислений. Каждое возможное направление

вычислений называется ветвью. Логическое условие называют простым, если разветвляющийся процесс имеет две ветви, и сложным, если процесс разветвляется на три и более ветви. Для выбора дальнейшей цепочки операций предназначен символ решения. Пример изображения данного элемента представлен на рис.2.1.

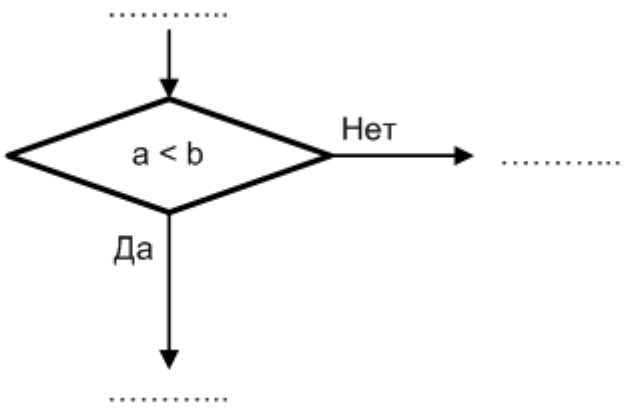


Рис.2.1. Пример изображения символа решения

Приведенный выше символ эквивалентен характерному практически для всех языков программирования условному оператору.

При использовании символа необходимо придерживаться следующих правил.

1. Символ может иметь один и только один вход и ряд взаимоисключающих (альтернативных) выходов.

2. Текст символа должен содержать описание какого-либо логического условия или действия, предполагающего сравнение элемента с набором допустимых значений. В случае задания логического условия допускается постановка знака вопроса (?). Размещение описания операций внутри символа не допускается.

3. Входным считается верхний угол блока, а выходными - остальные.

4. Выходящие ветви должны сопровождаться текстом, однозначно определяющим выполняемые в дальнейшем операции (выбранный путь). В случае, когда текст символа содержит условие,

разрешается использовать пары идентификаторов вида "Да"-"Нет", "1"-"0", "Истина"-"Ложь", "True"-"False" и т.д.

Как было сказано ранее, символ решения может иметь более двух выходов. В этом случае допускается наличие как нескольких выходящих ветвей, так и одной ветви, разделяющейся в дальнейшем. Все прочие рекомендации по оформлению символа остаются актуальными. Пример изображения символа решения с группой выходов представлен на рис.2.2.

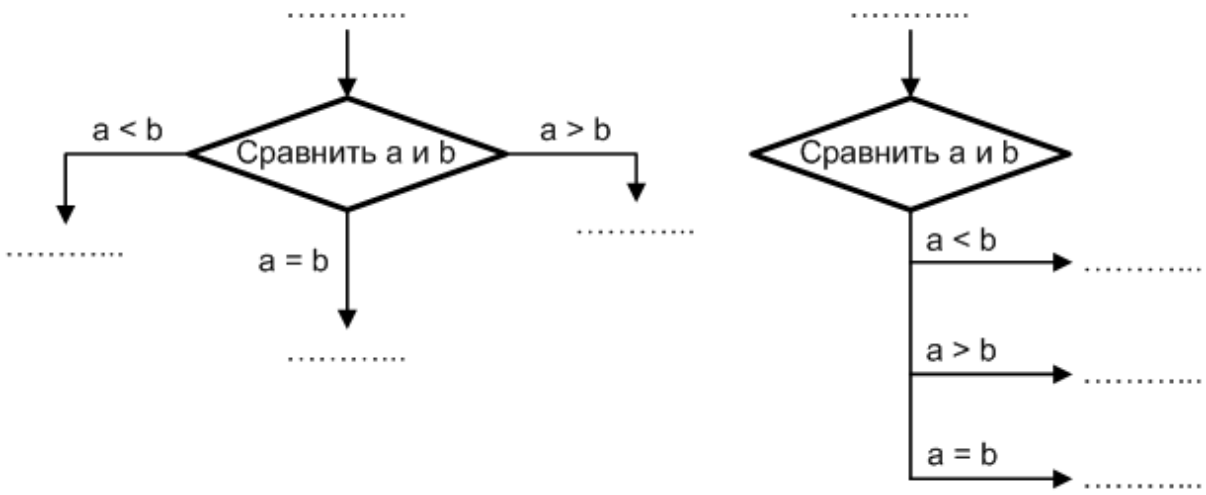


Рис.2.2. Пример изображения символа решения с группой выходов

Следует обратить внимание, на тот факт, что при наличии ровно двух альтернатив рекомендуется в целях внесения большей ясности избегать варианта оформления с единственным разделяющимся выходом.

## 2.2 Организация циклических процессов

Часто при решении задач приходится многократно выполнять одни и те же операции для различных значений входящих величин. Такие многократно повторяемые участки вычислительного процесса называются алгоритмами циклической структуры, или циклами. Использование циклов позволяет существенно сократить объем схемы алгоритма и длину соответствующей ей программы. Группы повторяющихся вычислений называют тело цикла. Специально изменяемый по заданному закону

параметр, входящий в тело цикла, называется переменной цикла. Переменная цикла используется для подготовки очередного повторения цикла и отслеживания условий его окончания.

В качестве переменной цикла используют любые переменные, индексы массивов, аргументы вычисляемых функций и т.д.

Для правильной организации цикла необходимо:

— задать перед циклом начальное значение переменной, изменяющейся в цикле;

— выполнить тело цикла;

— изменить переменную перед каждым новым повторением цикла;

— проверить условие окончания или повторения цикла.

Циклические алгоритмы, в которых число повторений заранее известно из исходных данных или определено в ходе решения задачи, называют детерминированными, если число повторений тела цикла заранее неизвестно, а зависит от значений параметров, участвующих в вычислениях цикл называют итерационным.

Для отображения циклических процессов используется символ, состоящий из двух частей, определяющий начало и конец цикла (рис.2.3). Обе части символа имеют один и тот же идентификатор. Условия для инициализации, завершения могут помещаться внутри символа в начале или в конце. В зависимости от расположения операции проверки условия циклы могут быть организованы как циклы с предусловием или с постусловием.


Рис.2.3. Пример изображения символа цикла

### 2.2.1 Циклы с предусловием

Если в цикле с предусловием входящие в тело цикла команды могут не выполняться ни разу (если начальное значение параметра цикла удовлетворяет условию выхода из цикла, рис.2.4)


Рис.2.4. Пример изображения цикла с предусловием

### 2.2.2 Циклы с постусловием

В цикле с постусловием - выполняются как минимум один раз (даже если начальное значение параметра цикла удовлетворяет условию выхода из него, рис.2.5).


Рис.2.5. Пример изображения цикла с постусловием

Операция по модификации переменой цикла может быть определена как в теле цикла, так и в символе конца. Недопустима модификация переменной цикла в символе начала цикла.

## 2.3 Примеры схем алгоритмов с ветвлениями

Пусть требуется реализовать программу для определения знака заданного числа.

Схема алгоритма для решения предлагаемой задачи представлена на рис.2.6.

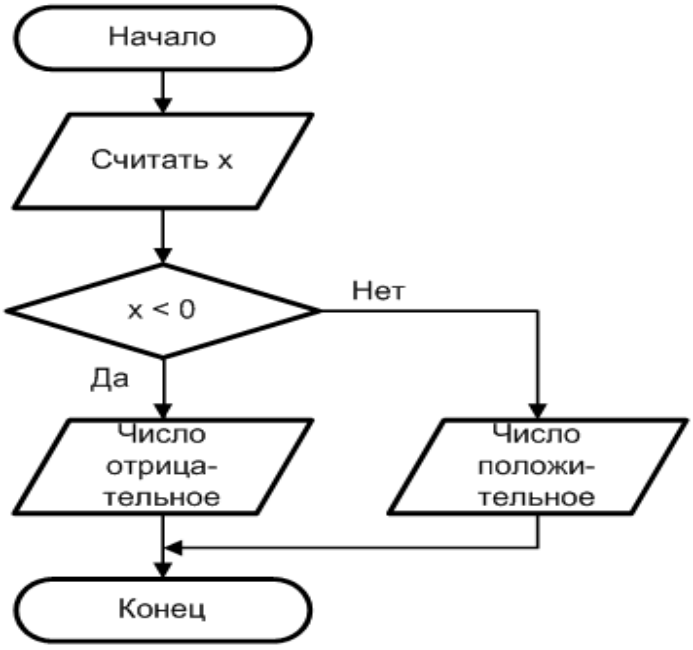


Рис.2.6. Схема алгоритма определения знака числа

Решение задачи на языке Pascal представлено ниже.

```
var x: real;
begin
  readln(x);
  if (x < 0) then
    writeln('Negative')
  else
    writeln('Positive');
end.
```

На языке программирования C:

```
#include <stdio.h>
int main() {
    float x;
    scanf("%f", &x);
    if(x < 0)
        printf("Negative\n");
    else
        printf("Positive\n");}
```

Пусть требуется реализовать программу для определения числа нулевых элементов в заданной матрице.

Схема алгоритма решения предлагаемой задачи представлена на рис.2.7.

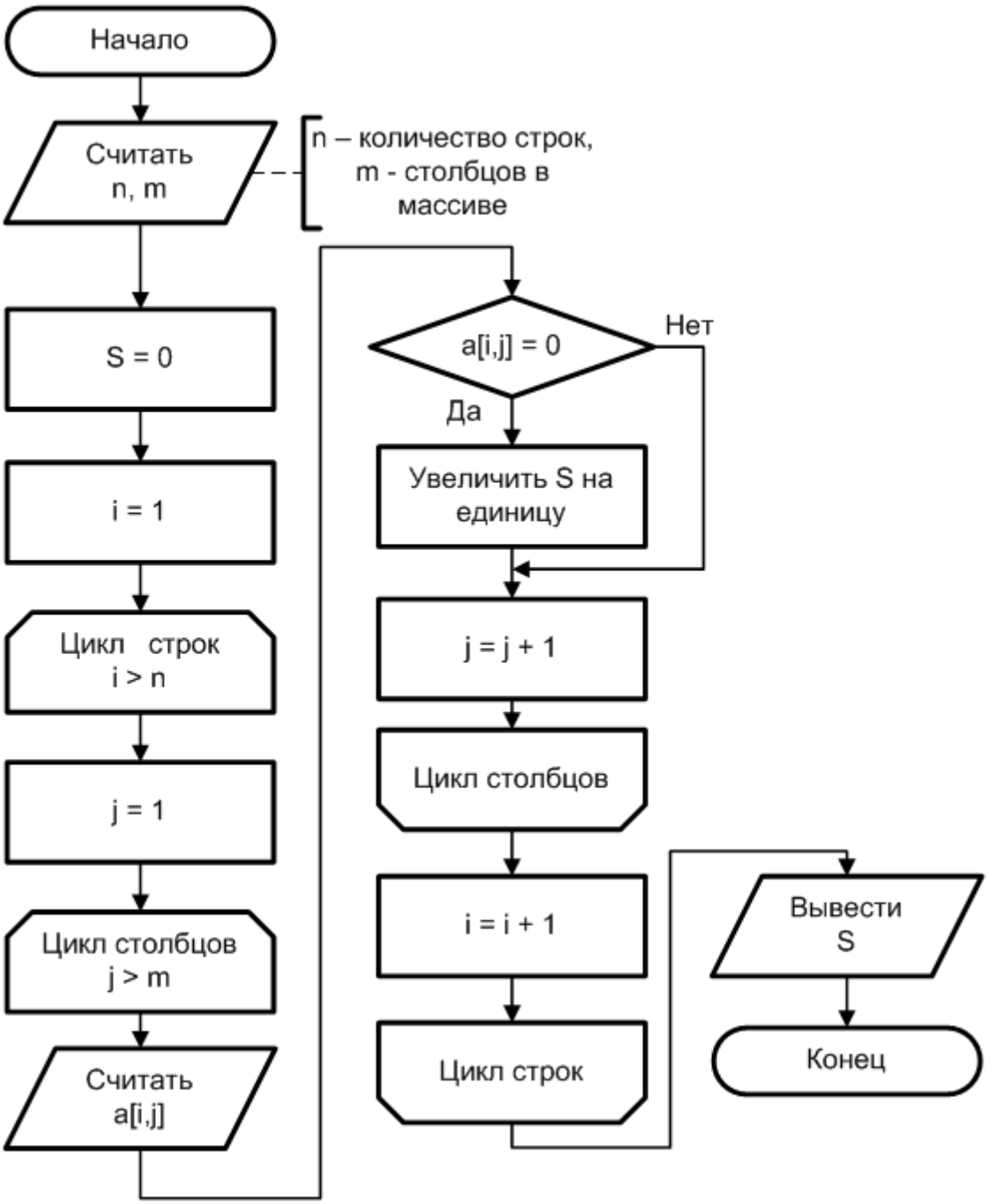


Рис.2.7. Схема алгоритма  вычисления суммы элементов

Эквивалентная программа на языке Pascal:

```
var n, m, i, j, s: integer;
    a: array [1..100, 1..100] of integer;
begin
  read(n, m);
```

```
  s := 0;
  writeln('Input matrix  ', n, ' to ', m ,' elements');
  for i := 1 to n do
    for j := 1 to m do
      begin
      read(a[i, j]);
      if a[i, j] = 0 then inc(s);
      end;
  writeln('Number of 0 elements ', s);
  readln;
end.
```

На языке программирования C:

```
#include <stdio.h>
int main() {
    int n, m, s = 0;
    int a[100][100];
    scanf("%d %d", &n, &m);
    printf("Input matrix %d  to  %d elements", n, m);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++) {
            scanf("%d", &a[i][j]);
            if(a[i][j] == 0)
                s++;
        }
    printf("Number of 0 elements %d", s);
}
```

Пусть требуется реализовать программу для вычисления суммы элементов заданного набора чисел. Мощность набора неизвестна заранее,

элементы вводятся в программу поочередно, последовательность заканчивается числом "0".

Схема алгоритма решения предлагаемой задачи представлена на рис.2.8.


Рис.2.8. Схема алгоритма вычисления суммы элементов

Программная реализация на языке Pascal:

```
var a, s: integer;
begin
  s := 0;
  repeat
    read(a);
    s := s + a;
```

```
  until a = 0;
  writeln('Summa of elements ', s);
  readln;
end.
```

На языке программирования C:

```
#include <stdio.h>
int main() {
   int a, s = 0;
   do {
      scanf("%d", &a);
      s = s + a;
   }
   while (a != 0);
   printf("Summa of elements %d", s);
}
```

## 2.4 Задачи для самостоятельной подготовки

Изобразите схемы алгоритмов для решения предлагаемых задач.

1. Выберите максимальное из двух заданных чисел.
2. По заданным параметрам a, b и c найдите корни квадратного уравнения.
3. Упорядочите три заданных числа по возрастанию.
4. Определите, является ли заданный год високосным.
5. Определите, сколько дней прошло между двумя датами.
6. Определите, какое число будет послезавтра.
7. Определите, принадлежит ли точка треугольнику, заданному координатами вершин.
8. Определите, расположен ли один прямоугольник внутри другого. Прямоугольники задаются координатами верхнего левого и правого нижнего углов.

9. Определите что больше: a в степени b или b в степени a?

10. Определите, может ли существовать треугольник с заданными длинами сторон.

11. Найдите сумму цифр заданного числа.

12. Найдите произведение цифр заданного числа.

13. Подсчитайте количество натуральных делителей заданного числа.

14. В заданном массиве найдите сумму максимального и минимального элементов.

15. В заданном массиве найдите k-ую статистику.

16. В заданном массиве подсчитайте сумму чисел, расположенных на нечетных позициях.

17. Реверсируйте заданную строчку.

18. Подсчитайте количество четных и нечетных элементов в массиве.

19. Найдите такие индексы i и j, что произведение соответствующих элементов массива минимально.

20. Подсчитайте сколько различных чисел присутствует в заданном наборе.

21. Определите, является ли заданный массив монотонным.

22. Проверьте, является ли заданное число простым.

23. Проверьте, является ли заданное число совершенным.

24. Проверьте, является ли строка палиндромом.

25. Найдите минимальный период в строке.

26. Определите, сколько символов необходимо дописать к строке с начала и с конца, чтобы она стала палиндромом.

27. Определите, является ли одна строка подстрокой другой.

28. Найдите максимальный элемент в заданной матрице.

29. Проверьте, является ли заданная двоичная матрица транзитивной.

30. Определите, имеются ли в заданной матрице одинаковые строки.

31. Найдите, если это возможно, произведение двух заданных матриц.

32. Найдите определитель матрицы 4x4.

33. Подсчитайте сумму элементов на диагоналях двумерного массива.

34. Определите что больше: a в степени b или b в степени a? Не используйте возведение в степень.

35. Определите, является ли заданный набор чисел монотонным. Не используйте массивы.

36. Найдите такие индексы i и j, что произведение соответствующих элементов массива минимально. Не используйте вложенные циклы.

37. Найдите минимальный период в массиве. Не используйте вложенные циклы.

38. Определите, сколько символов необходимо дописать к строке с начала и с конца, чтобы она стала палиндромом. Не используйте вложенные циклы.

39. Определите, какая из трехсимвольных подстрок встречается в заданной строке чаще всего.

40. Подсчитайте сумму элементов на диагоналях трехмерного массива.

# 3 Операторы переходов

В данном разделе рассматриваются вопросы, связанные с оформлением операторов управления потоком команд.

## 3.1 Оператор безусловного перехода

Безусловным принято называть принудительный переход к следующей выполняемой инструкции, отличной от очередной по порядку. В большинстве языков программирования, предоставляющих программисту возможность организации безусловного перехода, для данной цели используется оператор goto.

Необходимо отметить, что среди профессионалов в области компьютерных технологий бытует совершенно справедливое мнение о вредности оператора безусловного перехода. В первую очередь данная позиция обосновывается тезисом о нарушении парадигмы структурного программирования. Для подробного ознакомления с вопросом рекомендуется обратиться к статье Э. Дейкстры "Go To Statement Considered Harmful" и теореме Бема-Якопини. Общая рекомендация такова: если в сложившихся обстоятельствах возможно обойтись без применения безусловного перехода - откажитесь от его использования.

В тех же случаях, когда его применение по каким-либо причинам неизбежно, следует помнить о том, что этот оператор влияет только на поток управления и никоим образом не взаимодействует с данными. Следовательно, схема алгоритма не может содержать специального символа, отражающего наличие принудительного перехода. Для этих целей предназначены линии соединений.

## 3.2 Операторы расширенного управления циклами

Типичными инструкциями расширенного управления циклами являются операторы break и continue. Первый позволяет аварийно завершить выполнение цикла вне зависимости от значения условия завершения, второй предназначен для принудительного перехода к следующей итерации цикла.

Аналогичным с безусловным переходом образом данные операторы являются исключительно управляющими, поэтому на блок-схеме отображаются с помощью соединительных линий. В первом случае линию

следует объединять с выходом символа завершения цикла, во втором - с входом символа изменения значения управляющих переменных перед блоком завершения цикла.

## 3.3 Примеры схем алгоритмов с операторами перехода

Пусть требуется реализовать программу для определения наличия некоторого числа K в заданном массиве из N элементов.

Схема алгоритма  для решения предлагаемой задачи представлена на рис.3.1.

Эквивалентная схеме программа на языке Pascal:

```
var n, k, x, i: integer; flag: boolean;
begin
  read(n, k);
  flag := false;
  for i := 1 to n do
    begin
    read(x);
    if (x = k) then
      begin
              flag := true;       writeln('Found!');      break;
      end;
    end;
  if (not flag) then    writeln('Not found!');
end.
```

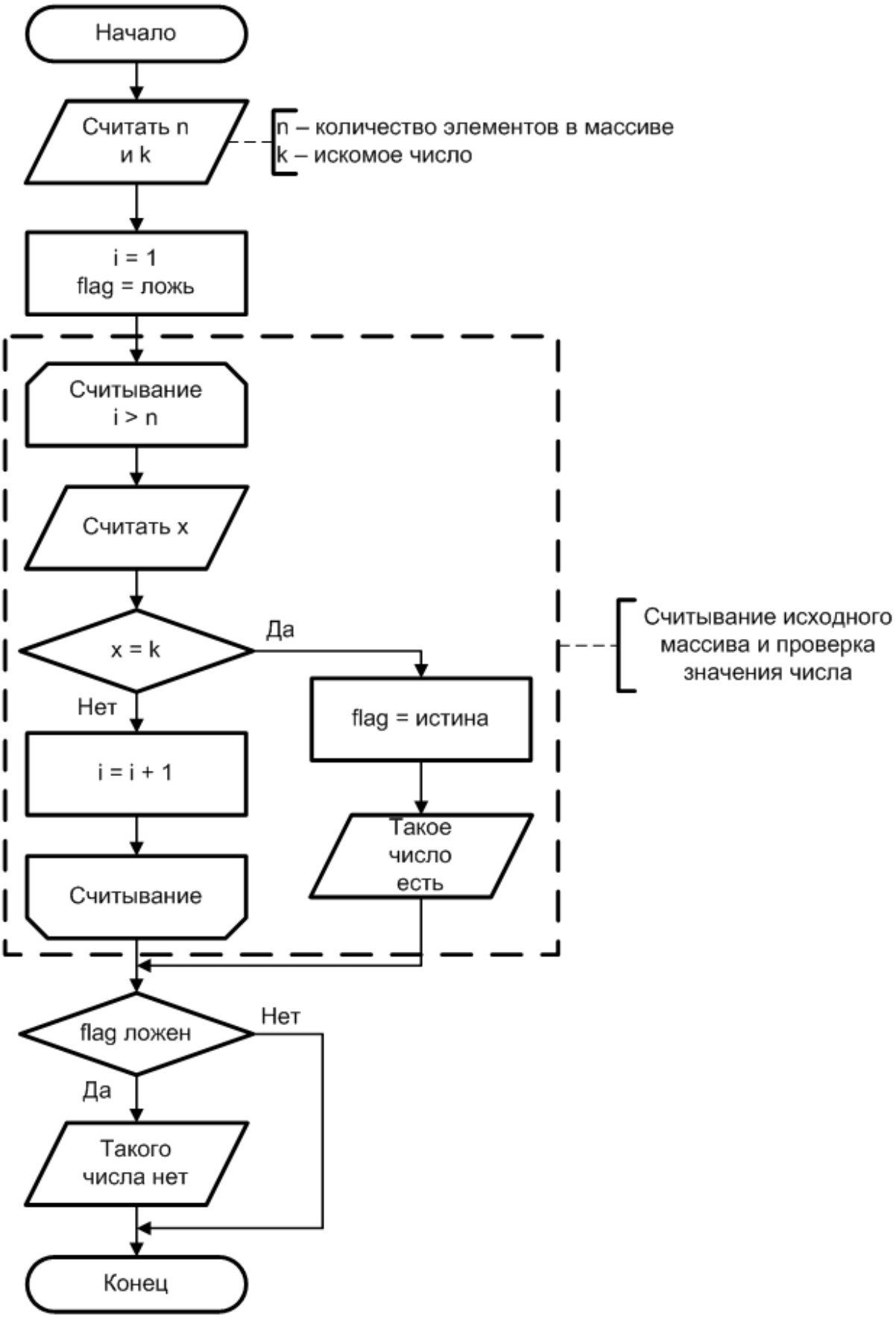


Рис.3.1. Схема алгоритма определения наличия числа в наборе

На языке программирования С:

```
#include <stdio.h>
int main() {
   int n, k, x, flag = 0;
   scanf("%d %d", &n, &k);
   for(int i = 0; i < n; i++) {
      scanf("%d", &x);
      if(x == k) {
         flag = 1;
         printf("Found!\n");
         break;
      }
   }
   if(flag == 0)
      printf("Not found!\n");
}
```

Пусть требуется реализовать программу для подсчета числа пар одинаковых элементов в массиве из N (известно, что N не превосходит 100) величин.

Схема алгоритма решения предлагаемой задачи представлена на рис.3.2.

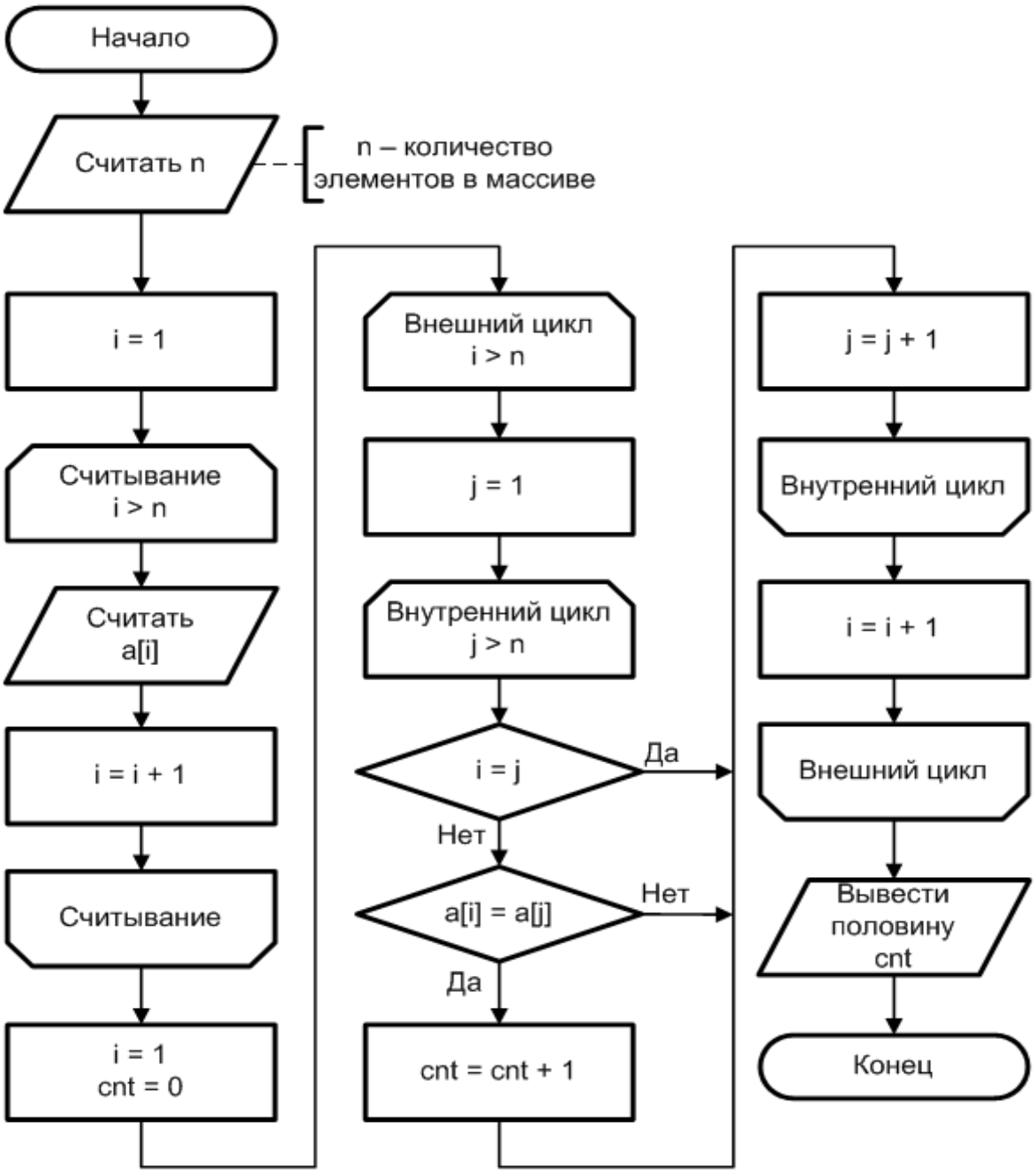


Рис.3.2. Схема алгоритма  подсчета числа одинаковых пар

Программа на языке Pascal:

```
var n, i, j, cnt: integer;
    a: array [1..100] of integer;
begin
  read(n);
  for i := 1 to n do
    read(a[i]);
```

```
  cnt := 0;
  for i := 1 to n do
    for j := 1 to n do
      begin
      if (i = j) then
        continue;
      if (a[i] = a[j]) then
        inc(cnt);
      end;
  writeln('Number of pairs is ', cnt div 2);
end.
```

На языке программирования C:

```
#include <stdio.h>
int main() {
    int n, cnt = 0;
    scanf("%d", &n);
    int a[n];
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++) {
            if(i == j)
                continue;
            if(a[i] == a[j])
                cnt++;
        }
    printf("Number of pairs is %d\n", cnt / 2);
}
```

## 3.4 Задачи для самостоятельной подготовки

Изобразите схемы алгоритмов для решения предлагаемых задач. Используйте операторы расширенного управления циклами.

1. Определите, присутствует ли в наборе чисел хотя бы одно отрицательное.
2. Определите, присутствует ли в заданном наборе чисел степень двойки.
3. Определите, лежат ли все точки из заданного набора на одной прямой.
4. Определите, присутствует ли в заданной строке подстрока из трех одинаковых букв.
5. Определите, является ли заданный массив строк упорядоченным.
6. Определите, является ли заданный массив знакочередующимся.
7. Подсчитайте количество строк в заданной матрице, таких, что в них есть хотя бы один нулевой элемент.
8. Найдите такое наименьшее K, что сумма первых K чисел заданного набора строго больше S.
9. Найдите первые сто простых чисел.
10. Найдите первые пять совершенных чисел.

# 4 Подпрограммы

В данном разделе приводятся рекомендации по оформлению составных частей программы - процедур и функций.

## 4.1 Понятие предопределенной процедуры

Оформление схемы сколь-нибудь больших алгоритмов практически невозможно без оперирования предопределенными процессами - последовательностями операций, определенных в другом месте (модуле, подпрограмме и т.д.). Символ предопределенного процесса изображен на рис.4.1.

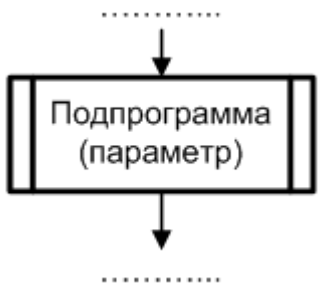


Рис.4.1. Пример изображения символа предопределенного процесса

Текст символа должен содержать текстовую метку (идентификатор), позволяющую однозначно идентифицировать набор инструкций, на который осуществляется ссылка. Если ссылка выполняется на отсутствующий в документации процесс, то рекомендуется явно указывать данный факт в комментариях. В остальном правила оформления данного символа не отличаются от правил оформления символа процесса.

Необходимо отметить, что применение рассматриваемого блока делает возможным декомпозицию разрабатываемого алгоритма на процедуры и функции.

## 4.2 Изображение процедур

Вызов процедуры из произвольного места алгоритма рекомендуется оформлять изображенным на рис.4.2 образом.

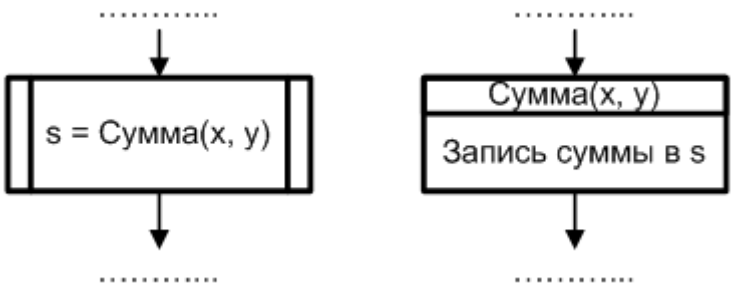


Рис.4.2. Пример изображения вызова процедуры

Следует обратить внимание на тот факт, что при вызове после идентификатора необходимо указывать передаваемые параметры. Тело процедуры рекомендуется оформлять согласно приведенному на рис.4.3 способу.

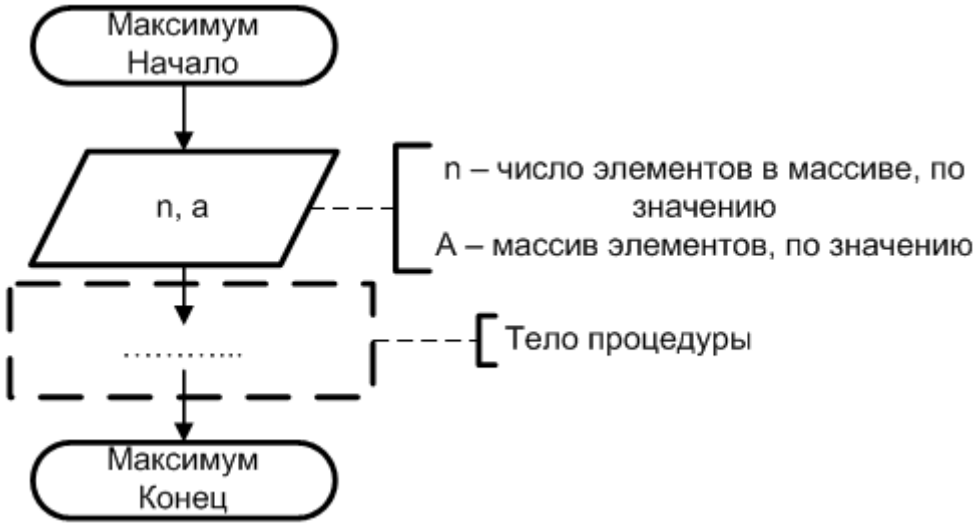


Рис.4.3. Пример изображения тела процедуры

Терминальные символы описания предопределенного процесса дополнительно к надписям "Начало" и "Конец" содержат присвоенную набору действий текстовую метку. В следующем за терминальным символом блоком ввода-вывода через запятую перечисляются параметры процедуры, причем в качестве имен применяются используемые в описании идентификаторы. В комментариях к символу ввода-вывода следует описать назначение принимаемых параметров и способ их передачи (по значению или по адресу).

## 4.3 Изображение функций

Вызов функции из произвольного места алгоритма оформляется аналогичным образом, за тем исключением, что возвращаемое значение может быть присвоено некоторому идентификатору. Пример вызова функции представлен на рис.4.4.

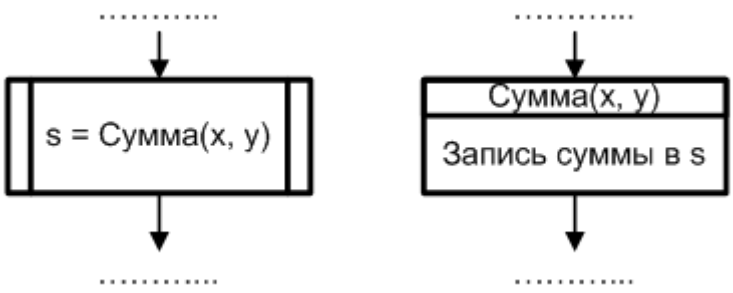


Рис.4.4. Пример изображения вызова функции

Точно таким же образом при вызове функции после идентификатора необходимо указывать передаваемые параметры. Тело функции оформляется согласно приведенному на рис.4.5 способу.

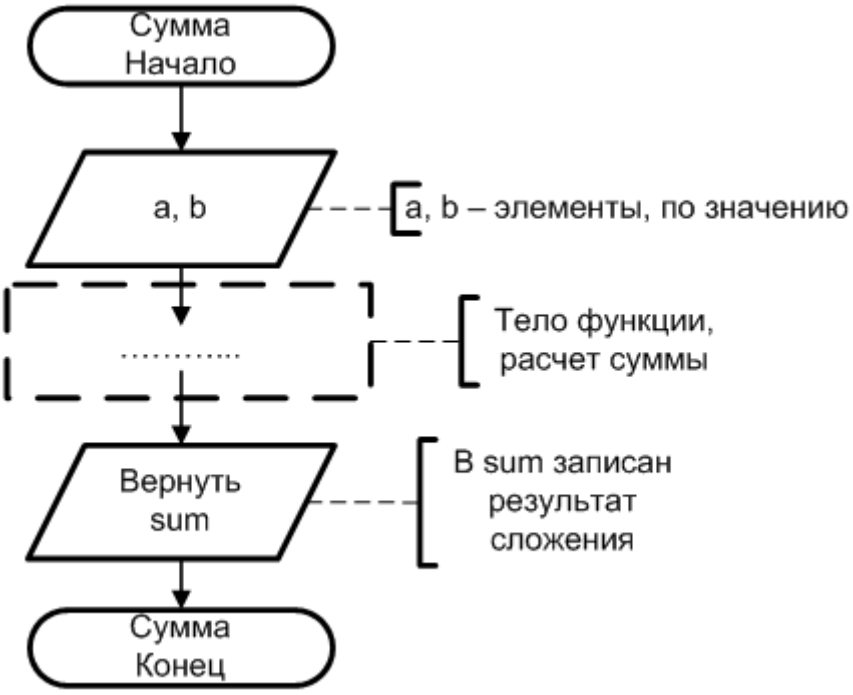


Рис.4.5. Пример изображения тела функции

Все перечисленные в предыдущем разделе положения об оформлении тела процесса остаются актуальными. Кроме того, перед завершающим терминальным символом необходимо явным образом указать возвращаемое функцией значение. В случае инвариантности допускается изображение нескольких символов с указанием возвращаемого значения. Однако структура предопределенного процесса должна быть выстроена таким образом, чтобы для любых входных данных происходило обращение ровно к одному возвращающему блоку ввода-вывода.

## 4.4 Примеры схем алгоритмов с использованием подпрограмм

Пусть требуется реализовать программу для упорядочивания элементов массива размерности N.

Схема алгоритма  для решения предлагаемой задачи представлена на рис.4.6.

В работе алгоритма используются предопределенные процессы "Сортировка" и "Больше", схемы которых изображены на рис.4.7 и 4.8 соответственно.

Наиболее важным элементом схемы на рис.4.7 является функция "Деление". Алгоритм реализации этой подпрограммы приведен на рис.4.9.

Для обмена пары элементов местами используется процесс "Обмен", схема которого представлена на рис.4.10.

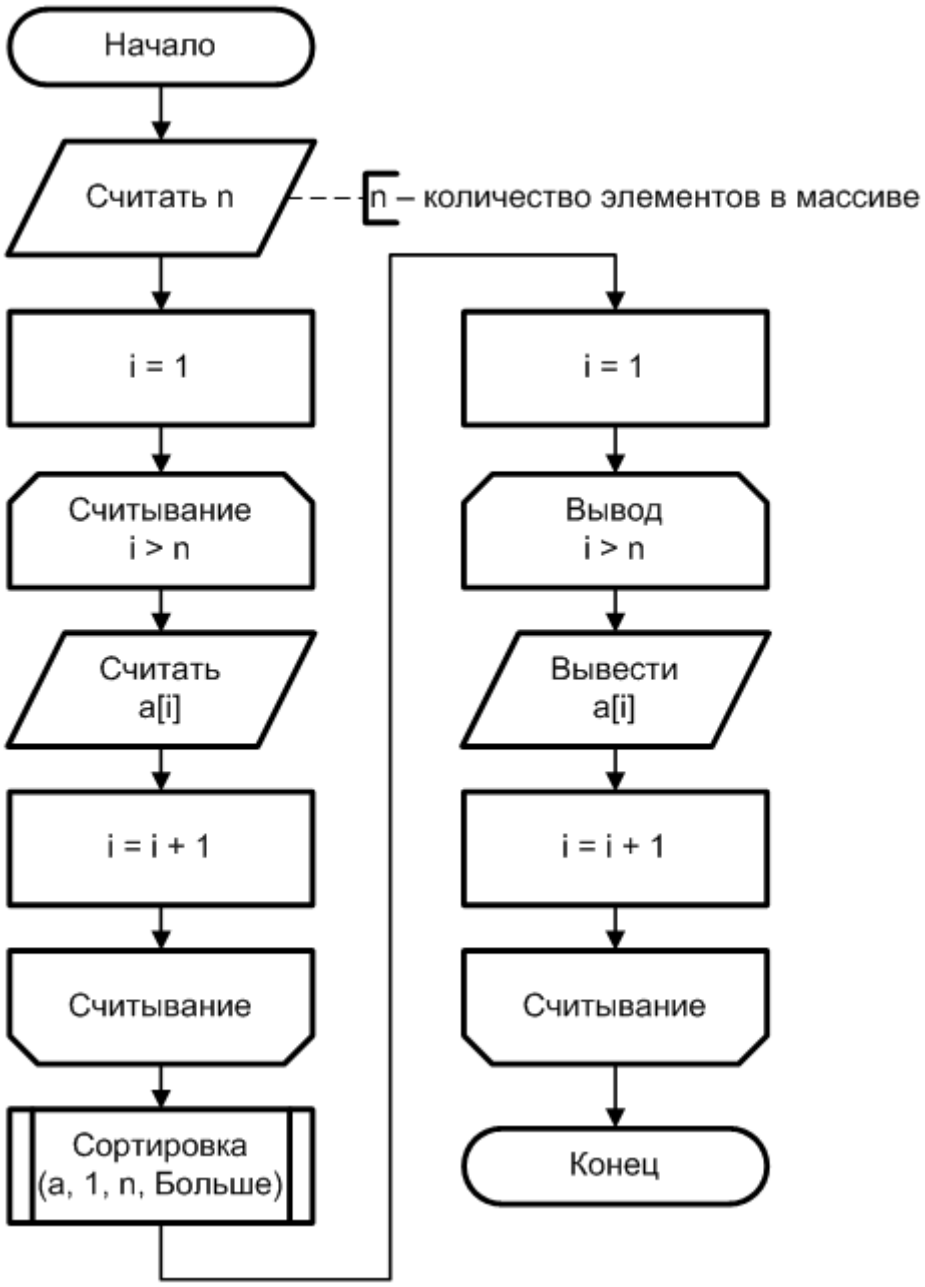


Рис.4.6. Схема алгоритма сортировки массива


Рис.4.7. Схема алгоритма предопределенного процесса "Сортировка"

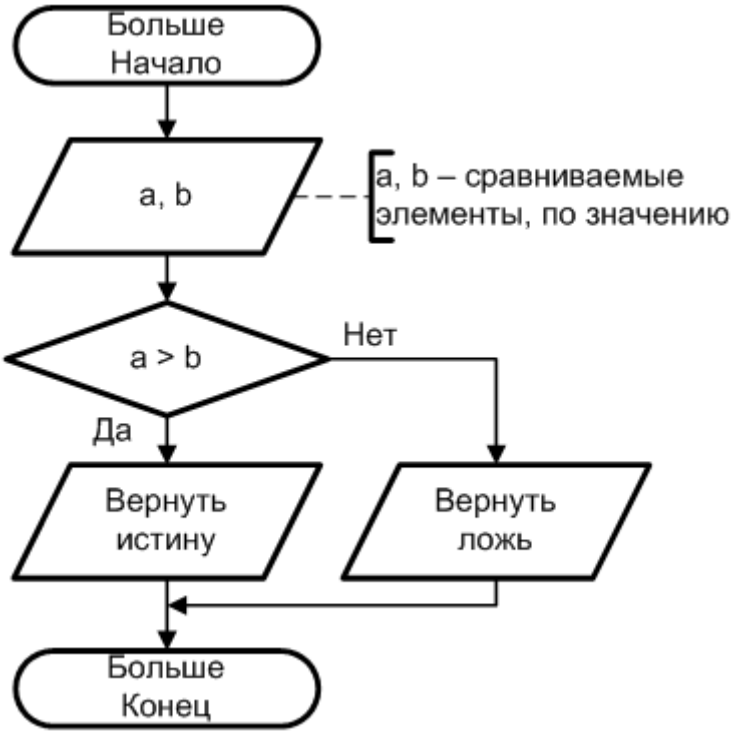


Рис.4.8. Схема алгоритма предопределенного процессе "Больше"

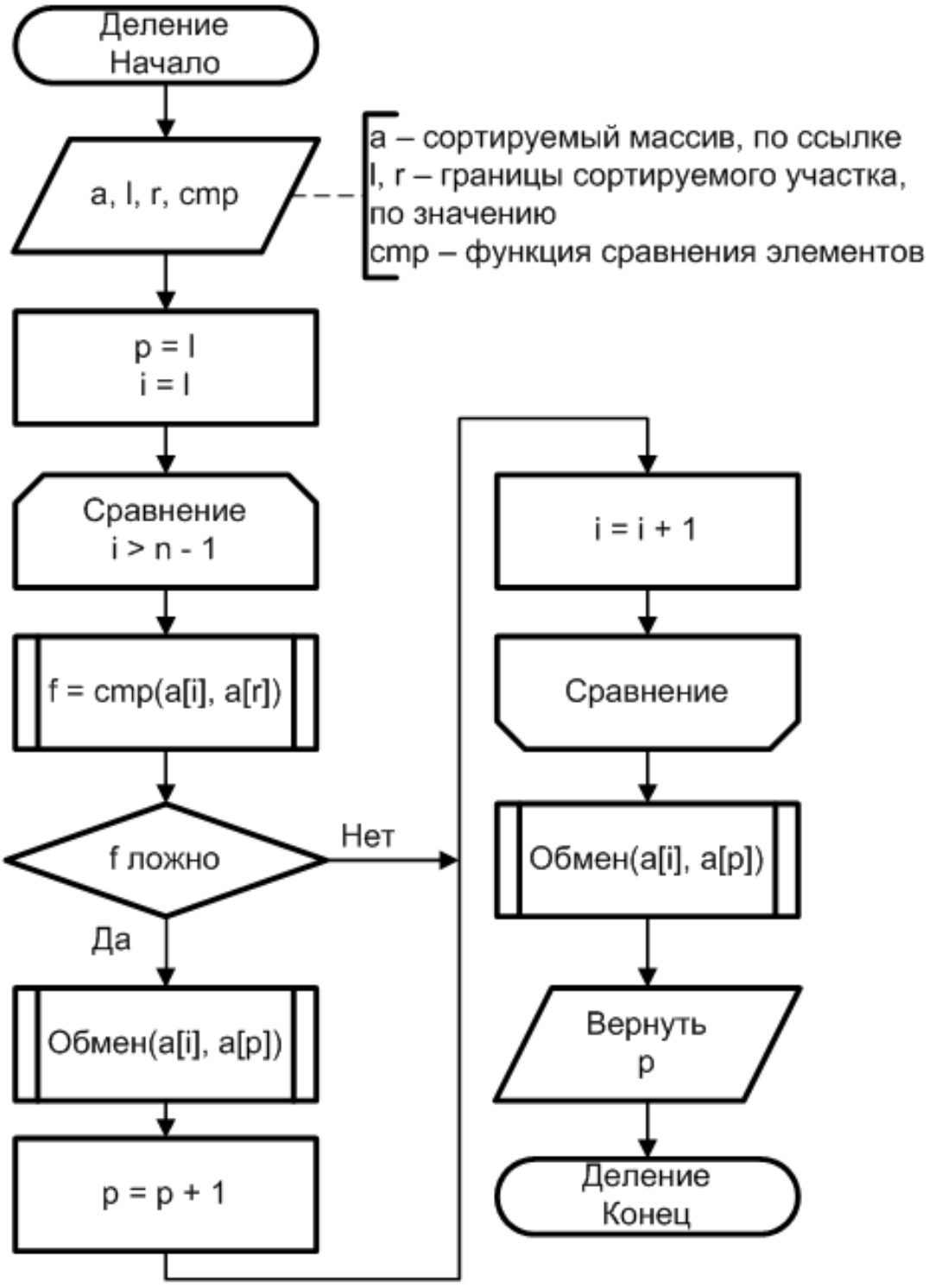


Рис.4.9. Схема алгоритма предопределенного процессе "Деление"

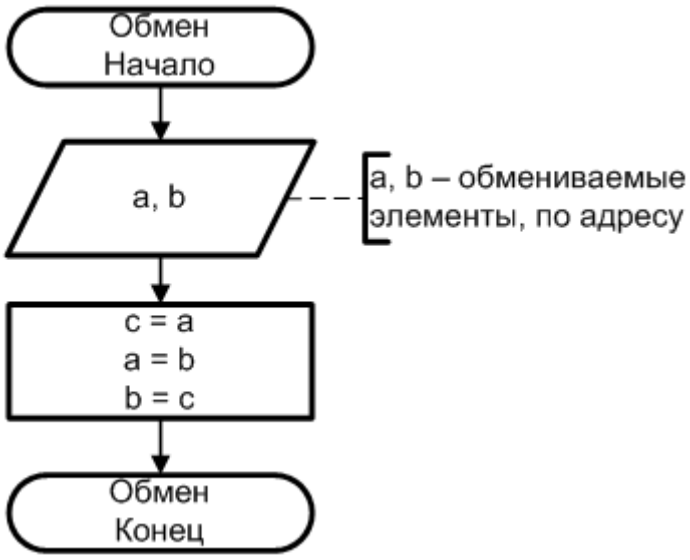


Рис.4.10. Схема алгоритма предопределенного процессе "Обмен"

Нетрудно заметить, что решение поставленной задачи основывается на рекурсивном алгоритмы быстрой сортировки. Код эквивалентной программы на языке Pascal может выглядеть следующим образом.

```
const maxsize = 1000;
type arrint = array [1..maxsize] of integer;
     func = function(a, b: integer): boolean;
var n, i: integer;
    a: arrint;

procedure swap(var a, b: integer);
var c: integer;
begin
  c := a;
  a := b;
  b := c;
end;

function gr(a, b: integer): boolean; far;
begin
```

```
  if (a > b) then
    gr := true
    else
    gr := false;
end;

function partition(var a: arrint; l, r: integer; cmp: func): integer;
var i, p: integer;
begin
  p := l;
  for i := l to r - 1 do
     if (cmp(a[i], a[r])) then
       begin
       swap(a[i], a[p]);
       p := p + 1;
       end;
  swap(a[r], a[p]);
  partition := p;
end;

procedure sort(var a: arrint; l, r: integer; cmp: func);
var q: integer;
begin
  if (l < r) then
    begin
    q := partition(a, l, r, cmp);
    sort(a, l, q - 1, cmp);
    sort(a, q + 1, r, cmp);
    end;
end;
```

```
begin
  read(n);
  for i := 1 to n do
    read(a[i]);
  sort(a, 1, n, gr);
  for i := 1 to n do
    write(a[i], ' ');
end.
```

На языке программирования C:

```
#include <stdio.h>

int i, n;
int a[1000];

void swap(int *a, int *b) {
  int c = *a;
  *a = *b;
  *b = c;
  return;
}

int gr(int a, int b) {
  if(a > b)
    return 1;
  else
    return 0;
}

int partition(int *a, int l, int r, int (*cmp)(int a, int b)) {
```

```
  int i, p = l;
  for(i = l; i < r; i++)
    if(cmp(a[i], a[r]) == 1) {
      swap(&a[i], &a[p]);
      p++;
    }
  swap(&a[r], &a[p]);
  return p;
}

void sort(int *a, int l, int r, int (*cmp)(int a, int b)) {
  int q;
  if(l < r) {
    q = partition(a, l, r, cmp);
    sort(a, l, q - 1, cmp);
    sort(a, q + 1, r, cmp);
  }
}

int main() {
  int i, n;
  scanf("%d", &n);
  for(i = 0; i < n; i++)
    scanf("%d", &a[i]);
  sort(a, 0, n - 1, gr);
  for(i = 0; i < n; i++)
    printf("%d ", a[i]);
  return 0;
}
```

## 4.5 Задачи для самостоятельной подготовки

Изобразите схемы алгоритмов для решения предлагаемых задач. Используйте процедуры и функции.

1. Найдите все простые числа в диапазоне от 1 до N.
2. Определите, какие из чисел в наборе являются простыми.
3. Найдите все палиндромы между двумя четырехзначными числами a и b.
4. Подсчитайте, сколько палиндромов содержатся в качестве подстрок в заданной строке.
5. Подсчитайте, сколько различных палиндромов содержатся в качестве подстрок в заданной строке.
6. Подсчитайте, сколько различных подстрок содержится в заданной строке.
7. Найдите наибольшую подстроку, встречающуюся в каждой строке из заданного набора.
8. Найдите наибольшую общую подпоследовательность, встречающуюся в каждой строке из заданного набора.
9. Найдите лексикографически минимальную строчку из заданного набора.
10. Для каждой строки матрицы подсчитайте среднее арифметическое.
11. Для каждого столбца матрицы подсчитайте среднее геометрическое.
12. Подсчитайте сумму минимумов каждой строки заданной матрицы.
13. Подсчитайте сумму максимумов каждого столбца заданной матрицы.
14. Подсчитайте сумму элементов квадратной матрицы, расположенных выше главной диагонали.

15. Найдите среднее арифметическое массива, образованного из максимальных элементов параллельных основной диагоналей заданной квадратной матрицы.

16. Найдите среднее геометрическое массива, образованного из минимальных элементов параллельных побочной диагоналей заданной квадратной матрицы.

17. Определите, сколько различных прямых можно построить на точках из заданного набора.

18. Выберите из заданного набора точек две, таких что декартово расстояние между ними минимально.

19. Выберите из заданного набора точек две, таких что манхэттенское расстояние между ними минимально.

20. Выберите из заданного набора чисел два, таких что хэммингово расстояние между их двоичным представлением минимально.

21. Для каждого десятичного числа из заданного набора выведите его двоичное представление.

22. Для каждого десятичного числа из заданного набора выведите его шестнадцатеричное представление.

23. Для каждого десятичного числа из заданного набора выведите его восьмеричное представление.

24. Упорядочите заданный набор чисел по возрастанию суммы их цифр.

25. Упорядочите заданный набор чисел по убыванию произведения их цифр.

26. Найдите цифровой корень заданного числа.

27. Подсчитайте значение функции Эйлера от заданного числа.

28. Подсчитайте остаток от деления числа a в степени b на c.

29. Найдите наибольший общий делитель двух заданных чисел.

30. Найдите наименьшее общее кратное двух заданных чисел.

31. Найдите наименьшее общее кратное для чисел из заданного набора.

32. Определите, сколько полных квадратов содержится в заданном наборе чисел.

33. Найдите такую подпоследовательность подряд идущих элементов заданного набора, что их сумма максимальна. Используйте структуру данных "сумматор".

34. Найдите такую подпоследовательность подряд идущих элементов заданного набора, что их сумма максимальна. Используйте структуру данных "sqrt-декомпозиция".

35. Найдите такую подпоследовательность подряд идущих элементов заданного набора, что их сумма максимальна. Используйте структуру данных "стек".

36. Выберите из заданного набора двоичных строк две, таких что расстояние Левенштейна между ними минимально.

37. Выберите из заданного набора двоичных строк две, таких что расстояние Дамерау-Левенштейна между ними минимально.

38. Найдите все простые числа в диапазоне от 1 до N. Используйте решето Эратосфена на битовом сжатии.

39. Найдите все простые числа в диапазоне от 1 до N. Используйте решето Сундарама.

40. Найдите все простые числа в диапазоне от 1 до N. Используйте решето Аткина.

## Заключение

Для разработки программного обеспечения активно используется множество языков программирования. Решение большинства практических задач часто требует проектирования больших программ со сложной логической структурой, состоящих из сотен, тысяч и даже

десятков тысяч операторов. Создание таких программ - труднейший процесс, требующий привлечения всех способностей человека к абстрактному мышлению, поэтому для облегчения восприятия на начальном этапе проектирования ПО удобными могут оказаться графические формы представления - схемы алгоритмов. Язык схем помогает повысить структурность программы, легко усваивается человеком и становится эффективным инструментом общения специалистов как на этапе постановки задачи, так и в процессе проектирования.

## Список литературы


1. А.М. Епанишников, В.А. Епанишников Программирование в среде Turbo Pascal 7.0. – 4-е изд., испр. и дополн. – М:Диалог-МИФИ, 2002 -367с.
2. Брайн Керниган, Деннис Ритчи Язык программирования С.- 2 изд. – М:Вильямс,2013 – 304с.
3. ГОСТ 19.701 – 90 «Единая система программной документации. Схемы алгоритмов, программ и данных. Условные обозначения и правила выполнения»