Тема 2. Основы алгоритмизации инженерных задач

Вопросы для изучения

2.1 Понятие алгоритма
2.2 Свойства и способы описания алгоритма
2.3 Базовые структуры алгоритмов: линейная, разветвляющаяся, циклическая
2.4 Типовые приемы алгоритмизации
2.5 Единая система программной документации. Правила выполнения схем алгоритмов
2.6 Алгоритм структуры вложенных циклов

Этапы подготовки и решения задач на ПЭВМ

Постановка задачи:

- формулировка условия задачи и определение целей решения задачи;
- определение формы выдачи результатов;
- описание данных (их типов, диапазонов величин, структуры и т. п.).

Разработка математической модели:

- анализ методов решения;
- разработка математической модели.

Разработка алгоритма:

- выбор формы записи алгоритма (блок-схемы, псевдокод и др.);
- проектирование алгоритма.

Программирование запись алгоритма на выбранном языке программирования.

Тестирование и отладка:

- синтаксическая отладка;
- отладка логической структуры;
- тестовые расчеты и анализ результатов тестирования;
- совершенствование программы.

Анализ результатов решения задачи и уточнение и в случае необходимости математической модели с повторным выполнением этапов 2-5.

2.1 Понятие алгоритма

В основе решения любой задачи на ЭВМ лежит понятие **алгоритма.**

Термин алгоритм происходит от algorithmi – латинской формы написания имени выдающегося математика IX века Аль-Хорезми, который сформулировал правила выполнения арифметических операций.

Алгоритм — это краткое и точное описание шагов ведущих к решению задачи, записанный по определенным правилам, обеспечивающим однозначность его понимания и механического исполнения при всех допустимых значениях исходных данных

2.2 Свойства и способы описания алгоритма

При составлении алгоритмов следует учитывать ряд требований, выполнение которых приводит к формированию необходимых свойств:

- **массовость** - алгоритм применим к целому классу однотипных задач, исходные данные которых могут меняться в определенных пределах.
- **детерминированность** - процесс применения правил к исходным данным (ход решения задачи) определен однозначно.
- **дискретность** - путь решения задачи определен в виде последовательности шагов – четко разделенных друг от друга и только выполнив один шаг, можно приступить к следующему.
- **результативность** – процесс решения должен закончиться за конечное число шагов и привести к решению задачи
- **понятность** - алгоритм создается в расчете на определенного исполнителя, необходимо, чтобы он мог понять и выполнить каждый шаг алгоритма.

Виды алгоритмов :

- **механический алгоритм** (детерминированные, жесткие) механический алгоритм задает определенные действия, обозначая их в единственной и достоверной последовательности, обеспечивая тем самым однозначный результат, если выполняются те условия процесса, задачи, для которых разработан алгоритм.

- **гибкие алгоритмы**:

вероятностный алгоритм дает программу решения задачи несколькими путями или способами, приводящими к вероятному достижению результата.

эвристический алгоритм – это такой алгоритм, в котором достижение конечного результата однозначно не предопределено, так же как не обозначена вся последовательность действий.

На практике наиболее распространены следующие формы представления алгоритмов:

- словесная (записи на естественном языке);
- графическая (изображения из графических символов);
- псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- программная (тексты на языках программирования).

Словесный способ не имеет широкого распространения по следующим причинам:

- такие описания строго не формализуемы;
- страдают многословностью записей;
- допускают неоднозначность толкования отдельных предписаний.

Пример алгорима вычисления периметра ребер и площади граней куба
– ввести длину ребра куба с формы приложения;
– вычислить периметр ребер;
– вычислить площадь граней;
– вывести результат на форму приложения.

Графическое изображение алгоритма широко используется перед программированием задачи вследствие его наглядности и облегчает процесс написания программы, ее корректировки при возможных ошибках, осмысливание процесса обработки информации.

Графическое изображение представляется в виде структурной **блок-схемы** –в виде связанных между собой с помощью **линий перехода** (потока управления) **блоков** – графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

Схема алгоритма является самым наглядным способом представления алгоритма, при этом нет никаких ограничений на степень детализации.

Псевдокод используется для облегчения разработки программ.

Так же как и программа, псевдокод имеет все достоинства структурированной записи, поэтому алгоритм, написанный на псевдокоде, достаточно легко преобразовать в программный код.

Пример псевдокода

Начало
Ввод A, B
Если A≥B, то Z=A
Иначе Z=B
Конец Если
Вывод Z
Конец

Основными достоинствами псевдокода являются:

– близость к языкам программирования;

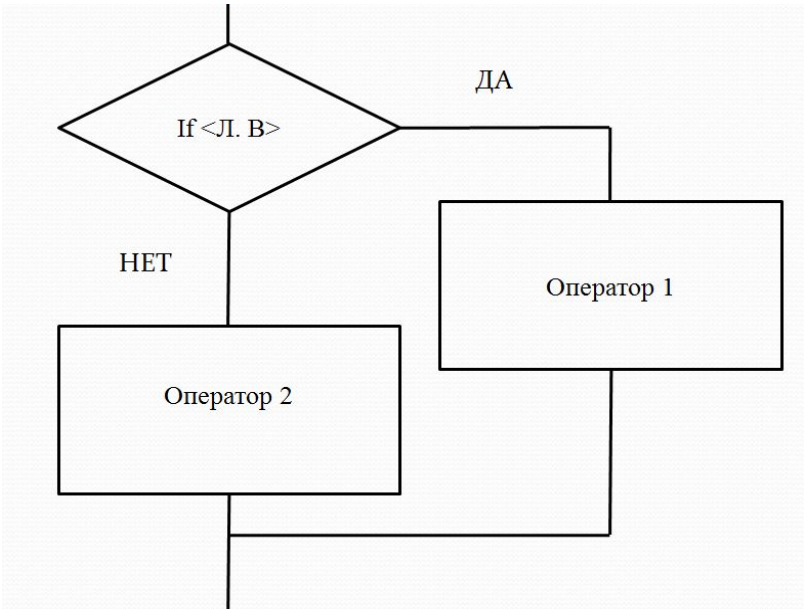
– возможность разобраться в самом длинном и сложном алгоритме, поэтому псевдокод используется чаще всего для документирования программ.

2.3 Базовые структуры алгоритмов: линейная, разветвляющаяся, циклическая

Линейный алгоритм – набор команд, выполняемых последовательно во времени друг за другом (рисунок 2.2)

Рисунок 2.2- Линейный алгоритм

Разветвляющийся алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов (рисунок 2.3)



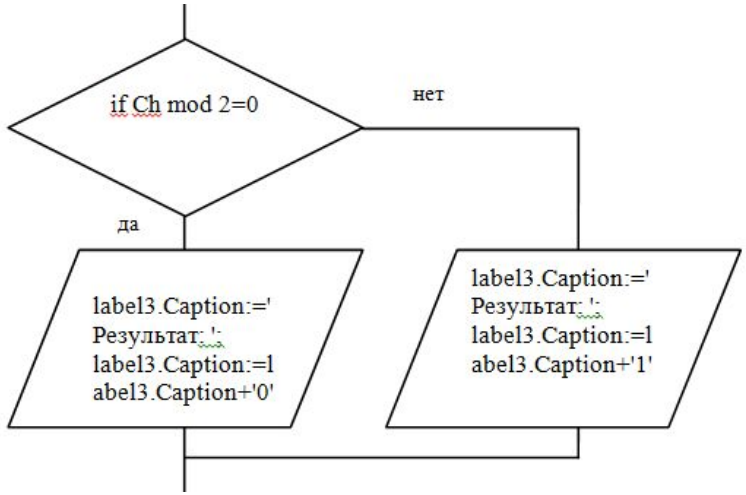


Рисунок 2.3- Разветвляющийся алгоритм

Условие – это логическое выражение, которое может принимать два значения – «ДА» (истина) или «Нет» (ложь). Если условие верно, действие выполняется, в противном случае – действие не выполняется.

Если имеется выбор нескольких альтернативных решений, то в схеме нужно предусмотреть несколько выходов. Это можно выполнить несколькими линиями от данного символа к другим, либо одной линией, которая имеет число разветвлений, соответствующих числу выходов.

При выполнении алгоритма после вычисления условий, записанных внутри символа Решение, один из выходов будет активизирован.

Соответствующие результаты вычисления могут быть записаны рядом с линиями, отображающими эти пути.

Циклический алгоритм – алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов. Многократно повторяющиеся участки называются циклами или телом цикла.

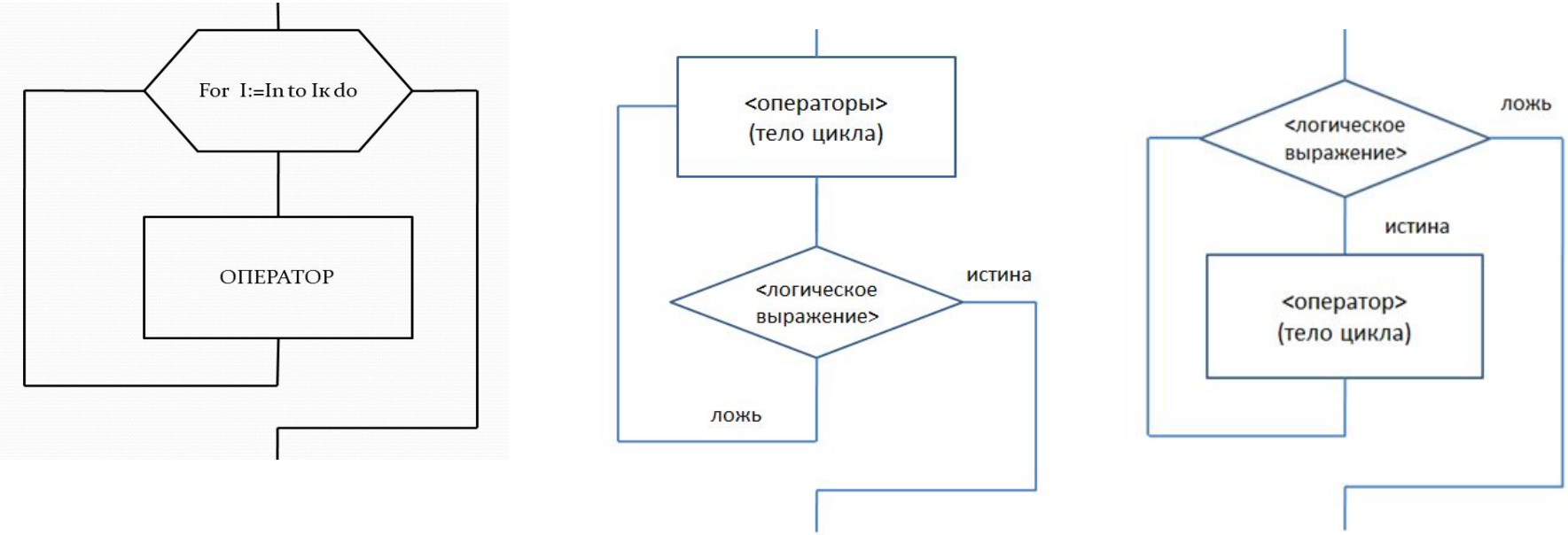


Рисунок 2.4- Циклические алгоритмы

Переменная алгоритма, которая при каждом выполнении цикла принимает новое значение, называется параметром цикла (илипеременной цикла).

Для организации любого цикла необходимо выполнение следующих условий:

– задание начального значения параметра (переменной) цикла перед началом цикла;

– изменение параметра (переменной) цикла перед каждым новым повторением тела цикла;

– проверка условия окончания (выхода из цикла) или повторения цикла;

– переход к началу цикла, если цикл не закончен, или выход из цикла, если условие выхода выполнено.

По месту расположения условий проверки повторения или окончания цикла можно выделить циклы с предусловием и постусловием.

По способу контроля окончания цикла различают следующие типы циклов.

1. **Количество повторений цикла неизвестно** (цикл с неизвестным числом итераций). Выход из цикла выполняется по дополнительному условию.

В инженерной практике при использовании численных методов часто применяется разновидность циклов с неизвестным числом повторений, которые называются итерационными. В итерационных циклах может отсутствовать переменная цикла. В итерационных циклах выполняется закон

2. **Тип арифметической прогрессии (цикл с известным числом итераций)**. В этих циклах параметр (переменная цикла) изменяется от заданного начального до заданного конечного значения, получая при каждом выполнении цикла постоянное приращение, которое называется шагом параметра цикла. Другоеназвание этого типа – циклы с параметром.

3. Алгоритмы решения сложных задач могут включать все перечисленные структуры и их комбинации.

Условия правильного использования циклов.

Начальное, конечное значения и шаг должны иметь тот же тип, что и параметр цикла.

Если начальное и конечное значения равны, то цикл выполняется один раз.

Внутри цикла не рекомендуется изменять параметр цикла и его конечное значение, т.к. эти значения устанавливаются в самом начале работы цикла.

Цикл заканчивается, когда параметр цикла принимает конечное значение.

При организации цикла следует особое внимание уделить правильному оформлению изменения параметра цикла, потому что ошибка на этом этапе может привести к «зацикливанию» вычислительного процесса.

2.4 Типовые приемы алгоритмизации

При решении большинства инженерных задач, встречающихся в практике, используется определенный набор типовых приемов алгоритмизации. Далее рассмотрим наиболее распространенные приемы.

Вычисление суммы. При вычислении суммы используется прием накопления. Вычисление суммы сводится к ее накоплению в виде значения переменной в цикле, в котором вычисляются соответствующие слагаемые. При этом вновь вычисленное слагаемое прибавляется к сумме предыдущих слагаемых, т.е. в цикле последовательно вычисляются все промежуточные суммы. Поэтому формула, предназначенная

для накопления суммы, имеет вид

$$S = S + y,$$

где y – очередное слагаемое;

S – промежуточная сумма.

После первого выполнения цикла первая промежуточная сумма должна быть равна значению первого слагаемого. Следовательно, начальное значение S должно быть равно нулю.

Вычисление произведения сводится к его накоплению в цикле в виде значения переменной, при этом в цикле вычисляются последовательно все промежуточные произведения. Формула для вычисления произведения имеет вид

$$P = P*y,$$

где y – очередной сомножитель;

P – промежуточное произведение.

Начальное значение P, которое задается перед циклом, должно быть равно единице.

Среди членов произведения не должно быть нулевых значений.

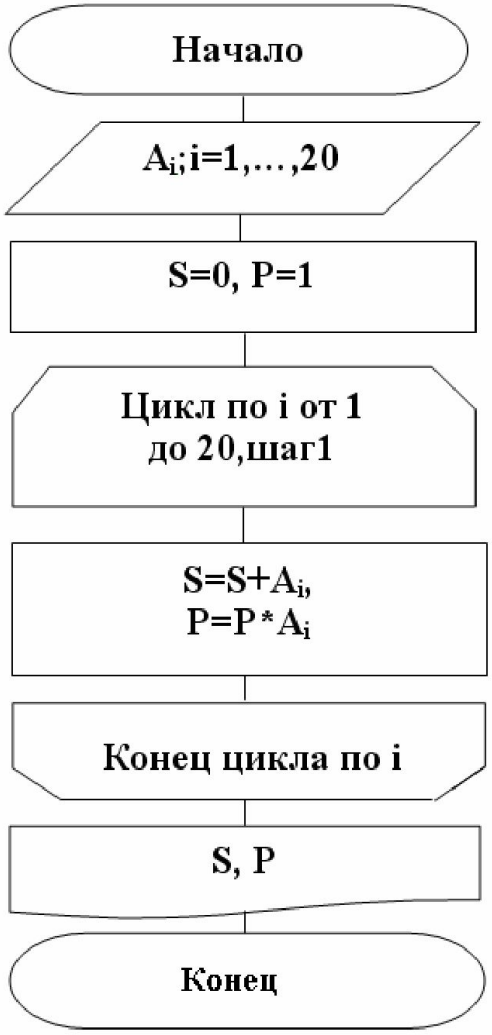
Задача.

Задан массив по имени A, состоящий из 20 элементов Ai, i = 1, …, 20.

Составить схему алгоритма вычисления суммы и произведения элементов этого массива.

Алгоритм вычисления будет состоять из следующих шагов.

1. Ввод массива Ai; i = 1, …, 20.
2. Задание начальных значений переменных S и P. S = 0, P = 1.
3. Организация цикла. Задаются параметры цикла: начальное значение параметра цикла 1, конечное значение 20, шаг 1.
4. Вычисление промежуточных значений S и P, т.е. накопление S и P. S = S + Ai; P = P*Ai.
5. Проверка окончания цикла. Если параметр цикла меньше конечного значения, то увеличиваем переменную цикла на шаг, т.е. на 1 и переходим к п. 4. Если переменная цикла больше конечного значения, то следующим выполняется п. 6.
6. Печать вычисленных значений S и P.
7. Конец.



Вычисление количества элементов. При вычислении количества элементов используется прием накопления, как и при вычислении суммы и произведения. Но в отличие от этих величин переменная, описывающая количество, должна иметь целый тип. Если не задано дополнительных условий, то правило изменения количества таково:

$$K = K + 1.$$

Начальное значение K = 0.

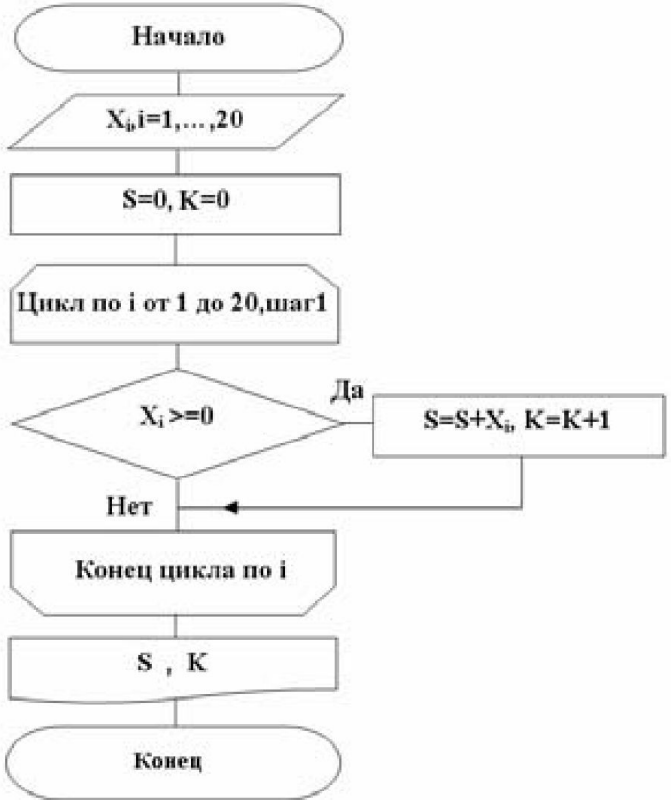
Задача.

Задан массив X, состоящий из 20 элементов, Xi, i = 1, …, 20.

Составить схему алгоритма вычисления суммы и количества положительных элементов массива.

Алгоритм вычисления будет состоять из следующих пунктов.

1. Ввод массива Xi; i = 1, …, 20.
2. Задание начальных значений переменных S = 0, K = 0.
3. Организация цикла. Задаются начальное и конечное значение переменной цикла и шаг цикла.
4. Проверка очередного элемента Xi на знак. Если условие Xi ≥ 0, «Да» (истина), то переход к п. 5, если «Нет» (ложь) – то к п. 6.
5. Накопленные суммы S = S + Xi и увеличение счетчика K на 1, K = K + 1.
6. Проверка окончания цикла.
7. Печать значений S и K.
8. Конец.



Нахождение максимального и минимального элементов в заданной последовательности

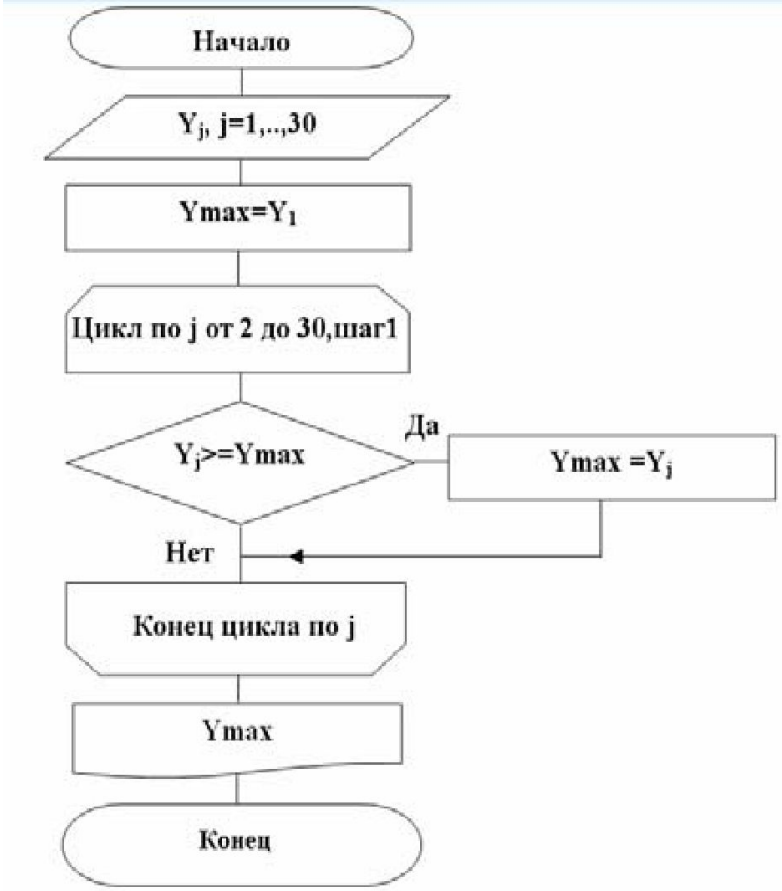
Задача. *Задан массив yj, j = 1, …, 30. Найти максимальный элемент этого массива.*

Поиск максимального (наибольшего) элемента массива выполняется в цикле путём последовательного сравнения значения текущего элемента массива с максимальным элементом из всех предыдущих. И если значение текущего элемента больше максимального из всех предыдущих, то максимуму присваивается значение текущего элемента.

Для применения указанного способа необходимо перед началом цикла задать начальное значение ymax, некоторый эталон переменной. Например, значение первого элемента массива. И поиск в цикле начинается со второго элемента. При первом выполнении цикла (j = 2) ymax будет сравниваться с y2. И если y2 будет больше ymax, то меняем эталон ymax, присваивая переменной ymax значения y2. И продолжаем сравнение, теперь уже со следующим элементом.

Алгоритм вычисления будет состоять из следующих пунктов.

1. Ввод исходного массива yj; j = 1, …, 30.
2. Задание начального значения ymax = y1.
3. Организация цикла. Задаются параметры цикла: начальное значение переменной цикла 2, конечное значение 30, шаг цикла 1.
4. Сравнение очередного j-го элемента yj и ymax. Проверяется условие yj ≥ ymax. Если условие выполняется, т.е. «Да», то переход на пункт 5, если «Нет», то – на конец цикла.
5. Присвоение ymax= yj.
6. Конец цикла.
7. Печать максимального элемента массива ymax.
8. Конец.



Задача. *Для массива Y, заданного в предыдущем примере, найти минимальный элемент Ymin и его порядковый номер (Jmin).*

Особенностью примера является то, что нужно найти не только минимальный элемент, но и его номер. Для решения данной задачи в алгоритм предыдущего примера нужно внести следующие изменения:

2. Задание начального значения Ymin = Y1 , Jmin = 1.

4. Сравнение очередного элемента Yj и Ymin. Если Yj меньше Ymin, то переход на п. 5, если больше – то на конец цикла.

5. Присвоение Ymin = Yj, Jmin = j.

7. Печать номера минимального элемента Jmin и значение минимального элемента Ymin.

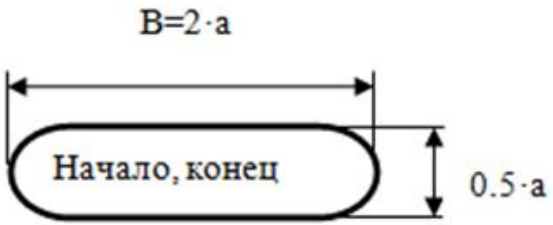
Так как цикл начнет выполняться со второго элемента массива, то может оказаться, что первый элемент Y1 будет минимальным. Поэтому в п. 2 задаётся начальное значение не только Ymin = Y1, но и Jmin = 1.

2.5 Единая система программной документации

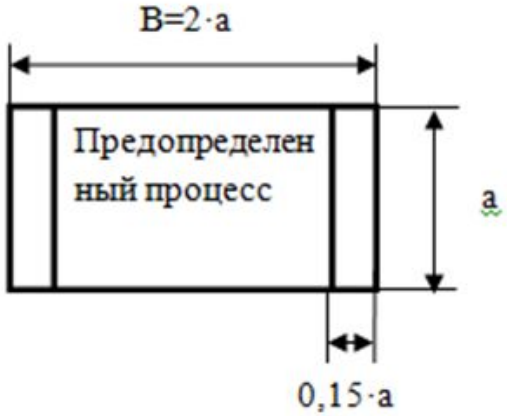
В схеме алгоритма каждому типу действий (например: ввод исходных данных, вычисление значений выражений, проверка условий и т.д.) соответствует геометрическая фигура, представленная символом действия (блоком). Блоки соединяют линиями переходов, которые определяют очерёдность выполнения действий.

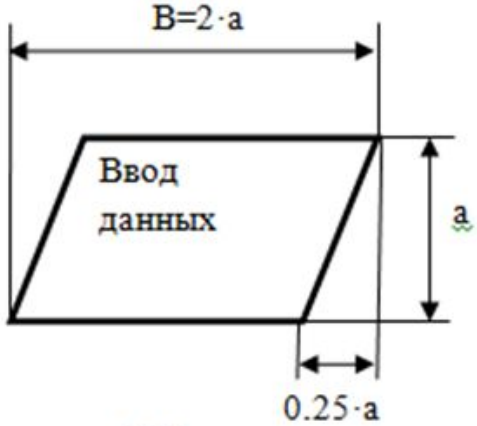
Форма блоков и правила составления схем установлены Единой Системой Программной Документации (ЕСПД) ГОСТ 19701-90.

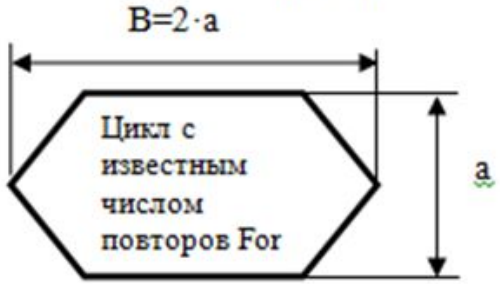
Наиболее часто употребляемые блоков указанного стандарта приведены на рисунке 2.1.

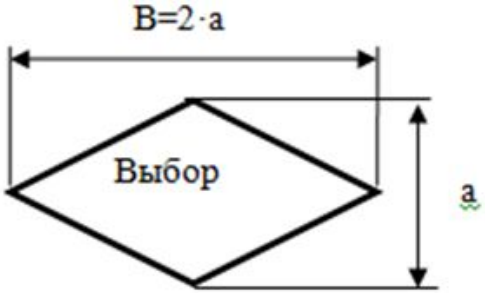

B=2·a
Начало, конец
0.5·a


B=2·a
Процесс(действие, операция)
a


B=2·a
Предопределенный процесс
a
0,15·a


B=2·a
Ввод данных
a
0.25·a


B=2·a
Цикл с известным числом повторов For
a


B=2·a
Выбор
a

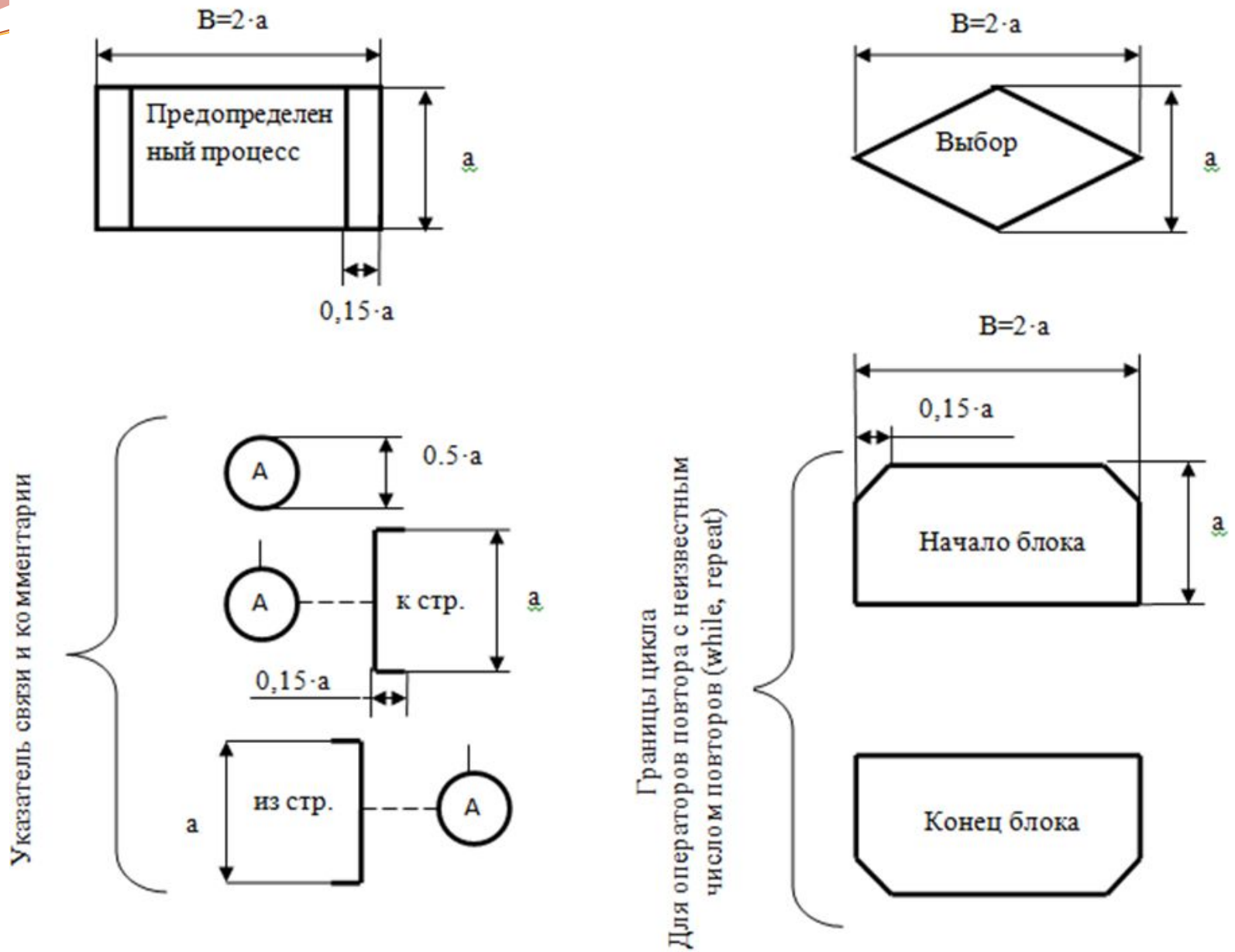


Рисунок 2.1- Блоки для построения блок-схем

Основные правила применения символов и выполнения схем алгоритмов:

1. Блоки в схеме должны быть расположены равномерно. Нужно придерживаться разумной длины соединений и минимального числа длинных линий.
2. Блоки должны быть по возможности одного размера и, предпочтительно, горизонтальной ориентации.
3. Внутри блока помещается минимальное количество текста, необходимое для понимания функции символа. Если объем текста превышает размер символа, то нужно использовать блок Комментарий.
4. Потоки данных и потоки управления в схемах показываются линиями. Направление потока слева направо и сверху вниз считается стандартным. Если поток имеет направление, отличное от стандартного, стрелки должны указывать это направление.
5. Линии в схемах должны подходить к блоку либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.
6. Каждый блок имеет один вход и один выход. Исключением является символ Решение (Выбор), который имеет один вход и несколько выходов. При этом каждый выход должен сопровождаться значениями условий

2.6 Алгоритм структуры вложенных циклов

Алгоритмы со структурой вложенных циклов. Любой цикл может содержать внутри себя один или несколько других циклов. Такая структура называется вложенными циклами. Охватывающие циклы называются внешними, охватываемые – внутренними. Вложенные циклы используются, например, для обработки матриц.

Параметры внешнего и внутреннего циклов должны быть разными. Они изменяются не одновременно, т.е. при одном значении параметра внешнего цикла параметр внутреннего цикла принимает поочередно все свои значения.

Программы решения многих задач требуют нескольких циклов.

Например:

– упорядочение массивов;

– обработка массивов;

– расчет таблицы значений функций, заданной степенным рядом.

В этих случаях важно правильно определить структуру алгоритма, прежде всего количество и относительное расположение циклов. В этих структурах могут использоваться рассмотренные приёмы алгоритмизации, но при этом необходимо определить, в каком цикле (внешнем или внутреннем) будет использоваться тот или иной приём.

Например, вычислить сумму всех элементов матрицы. В этом примере начальное значение суммы элементов нужно задать перед внешним циклом, а накапливать её во внутреннем цикле.

Если необходимо вычислить сумму элементов каждой строки матрицы, то начальное значение суммы нужно задать перед внутренним циклом, в котором перебираются и суммируются элементы одной строки матрицы. При этом внешним обязательно должен быть цикл, изменяющий номер строки, а внутренним – изменяющий номер столбца.

Задача. *Задана матрица X(N*M). Определить количество положительных элементов в каждой строке матрицы.*

Алгоритм решения состоит из следующих пунктов.

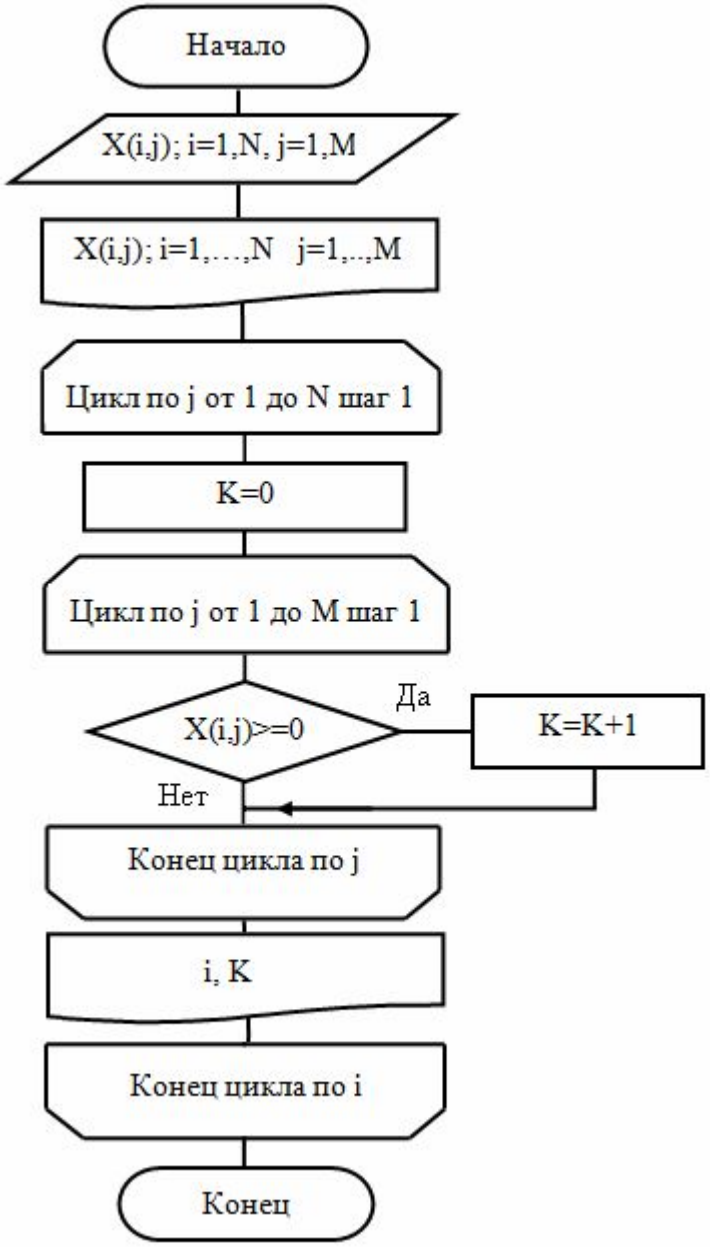
1. Ввод матрицы X(N, M), где i = 1, …, N; j = 1, …, M.

2. Печать матрицы X(N, M), где i = 1, …, N; j = 1, …, M.

3. Организация внешнего цикла по строкам. Переменная цикла i изменяется от 1 до N, шаг 1.

4. Задание начального значения счетчика количества положительных элементов для каждой строки матрицы K = 0.

5. Организация внутреннего цикла по столбцам. Параметр цикла j изменяется о 1 до M, шаг 1.



6. Проверка очередного элемента X(i, j) на знак. Если X(i, j) больше или равно нулю, то переход на п. 7, иначе – на п.8.

7. Увеличение счетчика количества положительных элементов K на 1, K = K + 1.

8. Конец внутреннего цикла по столбцам.

9. Печать номера строки i и количества положительных элементов K.

10. Конец внешнего цикла по строкам.

11. Конец.

Анализ данной схемы позволяет дать следующую рекомендацию: внутренний цикл необходимо переводить в исходное состояние непосредственно перед его началом (положение пункта обнуления счетчика количества положительных элементов).

Следующая рекомендация касается написания программ. Для наглядности вложенность циклов можно указывать отступами.