

**Дисциплина:**

**Проектирование и архитектура программных систем**

**Лабораторная работа №2:**

**Языки спецификаций передачи управления (алгоритм).**

**Блок-схемы. Схемы Несси-Шнейдермана.**

Преподаватель:

**Смирнов Константин Алексеевич**

**konst17@mail.ru**

**+79816807560**

## ***Понятие алгоритма***

Решение задачи при помощи ЭВМ начинается с составления алгоритма. Что же такое алгоритм?

Происхождение термина «алгоритм» связывают с именем великого математика Мухаммеда аль-Хорезми (763–850 гг.), который разработал правила выполнения четырех арифметических действий.

Согласно ГОСТ 19781-74:

Алгоритм – это точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату.

То есть ***алгоритм*** – это четкое указание исполнителю алгоритма выполнить определенную последовательность действий для решения поставленной задачи и получения результата.

Разработать алгоритм означает разбить задачу на определенную последовательность шагов. От разработчика алгоритма требуется знание особенностей и правил составления алгоритмов.

*Основные особенности алгоритмов:*

1. Наличие ***ввода*** исходных данных.
2. Наличие ***вывода*** результата выполнения алгоритма, поскольку цель выполнения алгоритма – получение результата, имеющего вполне определенное отношение к исходным данным.
3. Алгоритм должен иметь ***дискретную структуру***, т.е. алгоритм представляется в виде последовательности шагов, и выполнение каждого очередного шага начинается после завершения предыдущего.
4. ***Однозначность*** – каждый шаг алгоритма должен быть четко определен и не должен допускать произвольной трактовки исполнителем.
5. ***Конечность*** – исполнение алгоритма должно закончиться за конечное число шагов.

**6. Корректность** – алгоритм должен задавать правильное решение задачи.

**7. Массовость** (общность) – алгоритм разрабатывается для решения некоторого класса задач, различающихся исходными данными.

**8. Эффективность** – алгоритм должен выполняться за разумное конечное время. При этом выбирается наиболее простой и короткий способ решения задачи при соблюдении, естественно, всех ограничений и требований к алгоритму.

### **Способы записи алгоритмов**

Разработанный алгоритм может быть представлен несколькими способами:

- 1) на естественном языке (словесная запись алгоритма);
- 2) в виде блок-схем (графическая форма);
- 3) на языке программирования.

**Словесная запись алгоритма.** Словесная форма используется обычно для описания алгоритмов, предназначенных *исполнителю – человеку*. Команды записываются на обычном языке и выполняются по порядку. В командах могут использоваться формулы, специальные обозначения, но каждая команда должна быть понятна исполнителю. Естественный порядок команд может быть нарушен (если требуется, например, переход к предыдущей команде или требуется обойти очередную команду при каком-то условии), в этом случае команды можно нумеровать и указывать команду, к которой требуется перейти. Например, *перейти к п.3* или *повторить с п.4*.

**Графическая форма.** Алгоритмы представляются в виде блок-схем. Существуют специальные стандарты для построения блок-схем, где определяются графические изображения блоков. Команды алгоритмов записываются внутри блоков на обычном языке или с использованием математических формул. Блоки соединяются по определенным правилам линиями связи, которые показывают порядок выполнения команд.

**На языке программирования.** Если алгоритм разработан для решения задачи на ЭВМ, то для того, чтобы он мог выполняться *исполнителем* – ЭВМ, его необходимо записать на языке, понятном этому исполнителю. Для этого разработано множество языков программирования для решения задач разных классов. Запись алгоритма на языке программирования называется *программой*.

### **Представление алгоритмов в виде блок-схем**

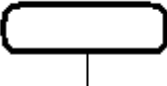
**Блок-схемой** будем называть такое графическое представление алгоритма, когда отдельные действия (или команды) представляются в виде геометрических фигур – *блоков*. Внутри блоков указывается информация о действиях, подлежащих выполнению. Связь между блоками изображают с помощью линий, называемых *линиями связи*, обозначающих передачу управления.

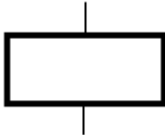
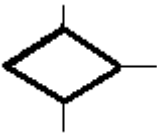
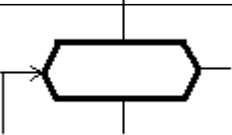
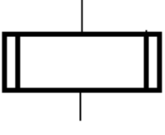
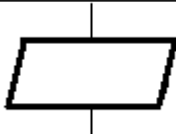
Существует Государственный стандарт, определяющий правила создания блок-схем. Конфигурация блоков, а также порядок графического оформления блок-схем регламентированы ГОСТ 19.701-90 "Схемы алгоритмов и программ". В табл. 2.1 приведены обозначения некоторых элементов, которых будет вполне достаточно для изображения алгоритмов при выполнении студенческих работ.

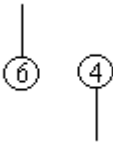

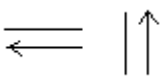
#### **Правила составления блок-схем:**

1. Каждая блок-схема должна иметь блок «Начало» и один блок «Конец».
2. «Начало» должно быть соединено с блоком «Конец» линиями потока по каждой из имеющихся на блок-схеме ветвей.
3. В блок-схеме не должно быть блоков, кроме блока «Конец», из которых не выходит линия потока, равно как и блоков, из которых управление передается «в никуда».
4. Блоки должны быть пронумерованы. *Нумерация* блоков осуществляется сверху вниз и слева направо, номер блока ставится вверху слева, в разрыве его начертания.

5. Блоки связываются между собой линиями потока, определяющими последовательность выполнения блоков. Линии потоков должны идти параллельно границам листа. *Если линии идут справа налево или снизу вверх, то стрелки в конце линии обязательны*, в противном случае их можно не ставить.
6. По отношению к блокам линии могут быть *входящими* и *выходящими*. Одна и та же линия потока является выходящей для одного блока и входящей для другого.
7. От блока «Начало» в отличие от всех остальных блоков линия потока только выходит, так как этот блок – первый в блок-схеме.
8. Блок «Конец» имеет только вход, так как это последний блок в блок-схеме.
9. Для простоты чтения желательно, чтобы линия потока входила в блок «Процесс» сверху, а выходила снизу.
10. Чтобы не загромождать блок-схему сложными пересекающимися линиями, линии потока можно разрывать. При этом в месте разрыва ставятся *соединители*, внутри которых указываются номера соединяемых блоков. В блок-схеме не должно быть разрывов, не помеченных соединителями.
11. Чтобы не загромождать блок, можно информацию о данных, об обозначениях переменных и т.п. размещать в *комментариях* к блоку.

Название блока	Обозначение блока	Назначение блока
1	2	3
Терминатор		Начало/Конец программы или подпрограммы

Процесс		Обработка данных (вычислительное действие или последовательность вычислительных действий)
Решение		Ветвление, выбор, проверка условия. В блоке указывается условие или вопрос, который определяет дальнейшее направление выполнения алгоритма
Подготовка		Заголовок счетного цикла
Предопределенный процесс		Обращение к процедуре
Данные		Ввод/Вывод данных

Соединитель		Маркировка разрыва линии потока
Комментарий		Используется для размещения пояснений к действиям
Горизонтальные и вертикальные потоки		Линии связей между блоками, направление потоков

## Типы алгоритмов

Тип алгоритма определяется характером решаемой в соответствии с его командами задачи. Различают три типа алгоритмов: линейные, разветвляющиеся, циклические.

**Линейный алгоритм** состоит из упорядоченной последовательности действий, не зависящей от значений исходных данных, при этом каждая команда выполняется только один раз строго после той команды, которая ей предшествует.

Таким, например, является алгоритм вычисления по простейшим безальтернативным формулам, не имеющий ограничений на значения входящих в эти формулы переменных. Как правило, линейные процессы являются составной частью более сложного алгоритма.



**Разветвляющимися** называются алгоритмы, в которых в зависимости от значения какого-то выражения или от выполнения некоторого логического условия дальнейшие действия могут производиться по одному из нескольких направлений.

Каждое из возможных направлений дальнейших действий называется *ветвью*.



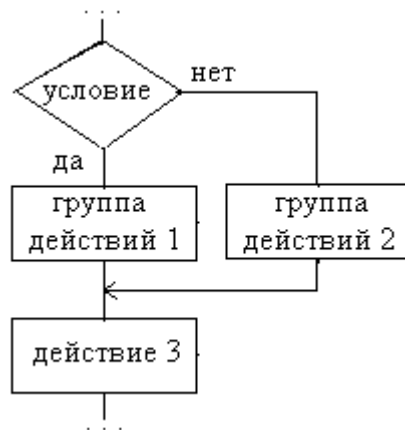
В блок-схемах разветвление реализуется специальным блоком «Решение». Этот блок предусматривает возможность двух выходов. В самом блоке «Решение» записывается логическое условие, от выполнения которого зависят дальнейшие действия.

Различают несколько видов разветвляющихся алгоритмов.

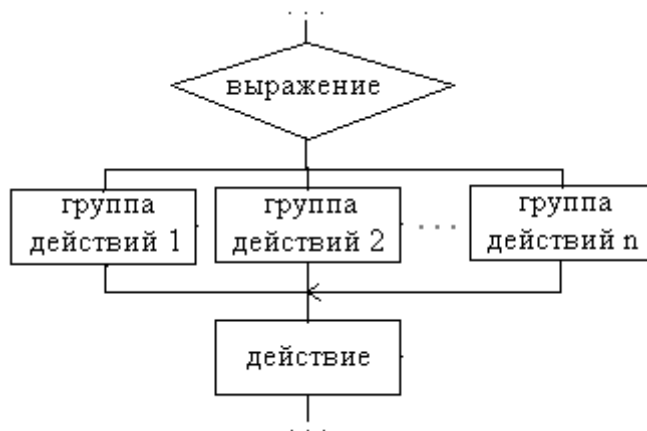
1. «*Обход*» – такое разветвление, когда одна из ветвей не содержит ни одного оператора, т.е. как бы обходит несколько действий другой ветви.



2. «*Разветвление*» – такой тип разветвления, когда в каждой из ветвей содержится некоторый набор действий.



3. **«Множественный выбор»** – особый тип разветвления, когда каждая из нескольких ветвей содержит некоторый набор действий. Выбор направления зависит от значения некоторого выражения.



**Циклические** алгоритмы применяются в тех случаях, когда требуется реализовать многократно повторяющиеся однотипные вычисления. *Цикл* – это последовательность действий, которая может выполняться многократно, т.е. более одного раза.

Различают:

- циклы с известным числом повторений (или со счетчиком);
- циклы с неизвестным числом повторений (циклы с предусловием и циклы с постусловием).

В любом цикле должна быть переменная, которая управляет выходом из цикла, т.е. определяет число повторений цикла.

Последовательность действий, которая должна выполняться на каждом *шаге цикла* (т.е. при каждом повторении цикла), называется *телом цикла* или *рабочей частью цикла*.

## Циклы со счетчиком

В циклах такого типа известно число повторений цикла, т.е. оно является фиксированным числом. В этом случае переменная, которая считает количество повторений (шагов) цикла, называется *счетчиком цикла* (или *параметром цикла*, или *управляющей переменной цикла*).

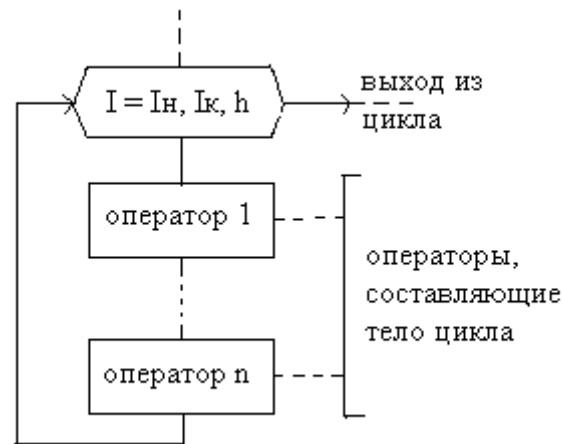
Блок-схема циклического алгоритма в этом случае выглядит так:



Такая блок-схема хорошо иллюстрирует работу цикла со счетчиком. Перед выполнением первого шага цикла счетчику цикла должно быть присвоено начальное значение – любое число в зависимости от алгоритма. Если величина счетчика цикла не превышает конечное значение, то далее будет выполняться группа действий, составляющих тело цикла. После выполнения тела цикла счетчик цикла изменяется на определенную величину – шаг изменения счетчика цикла  $h$ . Если полученное значение счетчика цикла не превысит конечное значение, то цикл продолжится

до тех пор, пока счетчик цикла не станет больше конечного значения – тогда управление передается действию, следующему за циклом.

В дальнейшем мы будем использовать в блок-схемах для изображения цикла со счетчиком блок «Подготовка». В блоке «Подготовка» записывается счетчик цикла ( $I$ ), далее последовательно указываются начальное значение ( $I_n$ ), конечное значение ( $I_k$ ) счетчика цикла и шаг его изменения ( $h$ ). Если шаг изменения  $h$  равен 1, его можно не записывать. Желательно, чтобы линия потока входила в блок сверху, линия потока к телу цикла выходила снизу, слева (или справа) входила линия потока перехода к следующему шагу цикла, а справа (или слева) выходила линия потока – выход из цикла.



При использовании цикла со счетчиком необходимо соблюдать некоторые требования:

- в теле цикла нельзя принудительно изменять значение счетчика цикла;
- не разрешается передавать управление *оператору тела цикла* извне, т.е. вход в цикл допускается только через начало цикла.

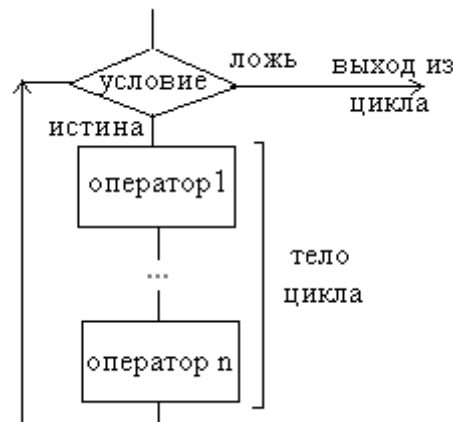
## Циклы с предусловием

Циклы с предусловием чаще всего используют тогда, когда неизвестно число повторений цикла. *Циклы с предусловием* – это такие циклы, в которых до начала выполнения тела цикла проверяется условие выполнения следующего шага цикла. Если значение этого условия истинно (т.е. условие выполняется), то выполняется тело цикла. В теле цикла должно изменяться значение по крайней мере одной переменной, которая влияет на значение условия (иначе произойдет «зацикливание»). Далее опять проверяется условие выполнения цикла, и если значение условия ложно, то осуществляется выход из цикла.

Можно использовать и еще один вариант этого цикла, когда проверяется не истинность значения условия, а ложность. В этом случае выход из цикла происходит, когда значение условия цикла становится истинным. Тот или иной вариант цикла используется в зависимости от того, какое условие в данном алгоритме программисту удобнее использовать.

*Особенность этого типа цикла в том, что тело цикла может не выполниться ни разу, если условие первоначально ложно в первом варианте (или истинно во втором).*

На блок-схеме такой цикл реализуется следующей конструкцией:



## Циклы с постусловием

Этот тип цикла также используется при неизвестном заранее количестве повторений цикла, но в отличие от цикла с предусловием здесь условие на выход из цикла проверяется после того, как выполнились операторы тела цикла, поэтому хотя бы один раз тело цикла будет обязательно выполнено.

На блок-схеме этот тип цикла изображается следующим образом:

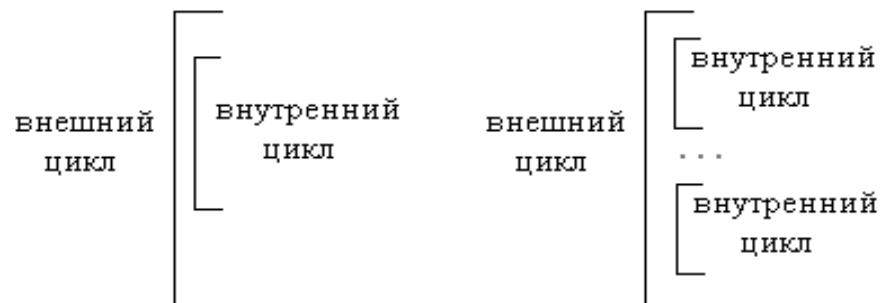


## Сложные циклы

**Циклы, которые содержат внутри себя (в теле цикла) один или несколько других циклов, называются сложными или вложенными циклами.**

При этом циклы, охватывающие другие циклы, называются *внешними*, а циклы, входящие во внешние, — *внутренними* циклами.

На каждом шаге внешнего цикла внутренний цикл «прокручивается» полностью.



## **Способы графического представления структурированных схем алгоритмов**

Многие специалисты в области теории программирования считают, что графическое представление алгоритмов в соответствии с *ГОСТ 19.701-90 (ISO 5807-85, см. выше)* скрывают структуру структурированной программы, представляя структурированность недостаточно очевидной. Поэтому для представления структурированных схем алгоритмов были предложены специальные методы графических обозначений, рассмотрим два из них – метод Дамке и диаграммы Насси-Шнейдермана.

**М. Дамке** предложил для конструкций структурированных схем алгоритмов специальные обозначения.

**Три основных конструкции** структурного программирования изображаются следующим образом.

1) **Функциональный блок** по-прежнему обозначается прямоугольником (рисунок ).

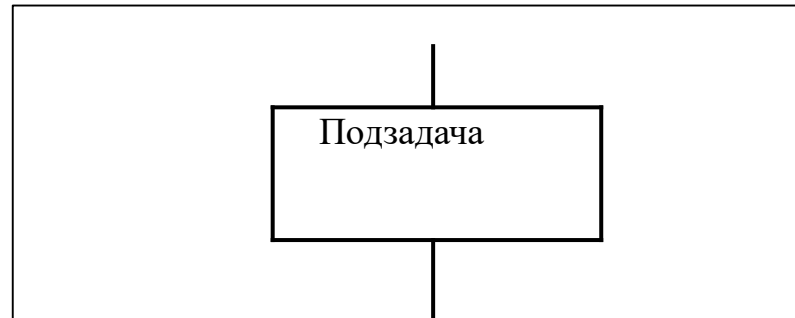


Рисунок – Представление функционального блока по методу Дамке

2) **Конструкция *If-Then-Else*** изображается так, как иллюстрирует рисунок 3.20.

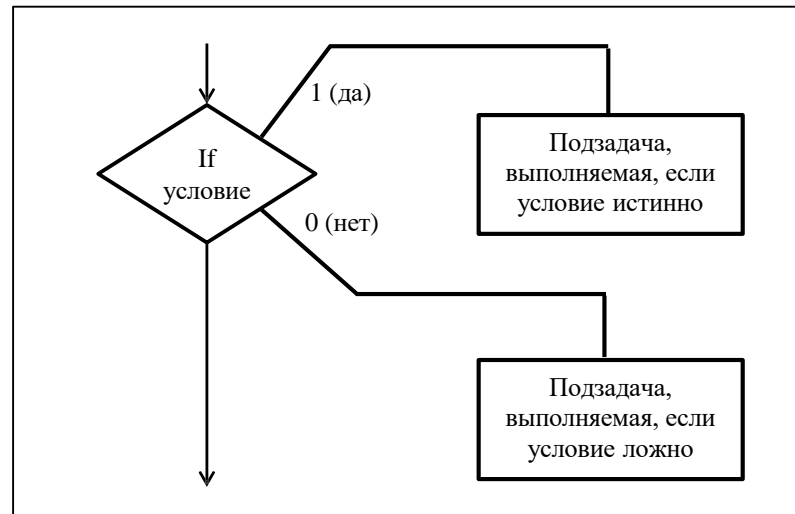
Элементы с выполняемыми действиями находятся справа от символа «Решение». Вход и выход из конструкции находятся соответственно сверху и снизу символа «Решение».

3) **Конструкция *Do-While* (цикл с предусловием “Пока”)** изображается так, как показывает рисунок 3.21.

Тело цикла выполняется до тех пор, пока условие истинно. Условие проверяется первым. Графически это изображается положением шестиугольника **над** выполняемым телом цикла.

Следует обратить внимание, что входы и выходы из всех конструкций метода Дамке находятся в левой части (сверху и снизу) графического представления конструкций. Расширения конструкций в правой части представления выходов не имеют.





Рисунок– Представление конструкции If-Then-Else по методу Дамке

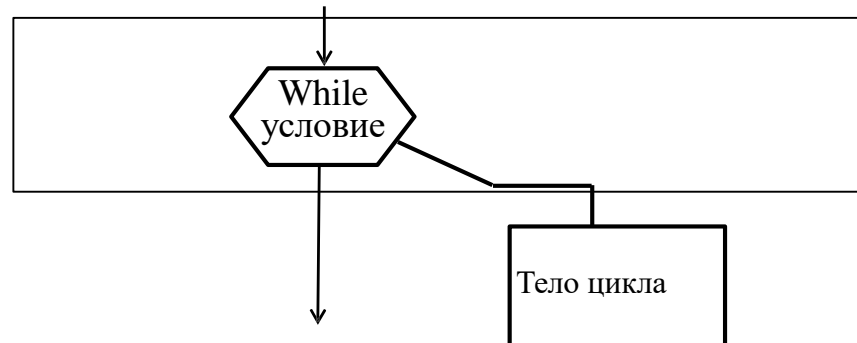
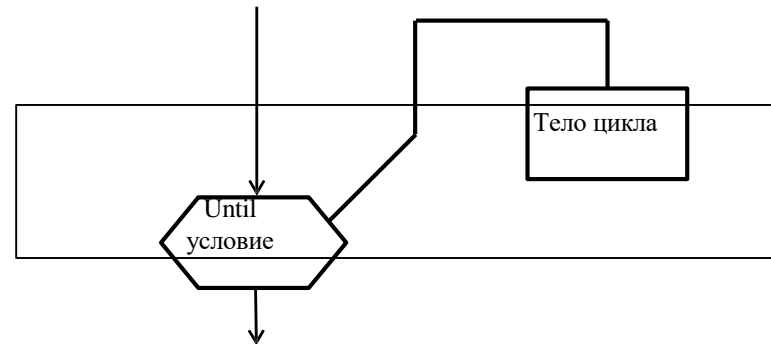


Рисунок – Представление конструкции Do-While по методу Дамке

Помимо трех основных конструкций в структурном программировании допускаются **дополнительные конструкции**. В методе Дамке они изображаются следующим образом.

1) **Конструкция Repeat-Until (цикл с постусловием “До”)** изображается так, как иллюстрирует рисунок 3.22.

Если условие истинно, осуществляется выход из цикла. Тело цикла выполняется до проверки условия. Графически это изображается положением шестиугольника **ПОД** телом цикла.



Рисунок– Представление конструкции Repeat-Until по методу Дамке

2) **Конструкция цикла с параметром** изображается так как представляет рисунок 3.23.

3) **Конструкция Case** изображается так, как иллюстрирует рисунок 3.24.

Основным принципом при разработке структурированных схем алгоритмов по методу Дамке является **принцип декомпозиции**. Он означает, что любой элемент, представляющий собой задачу, можно разделить на несколько элементов, образующих необходимые подзадачи.

Элементы в самой левой части схемы представляют укрупнённую структуру алгоритма. Затем элементы расширяются вправо по мере разделения каждого элемента на подзадачи.

Чтобы исследовать любую подзадачу, достаточно анализировать только те элементы и управляющие структуры, которые находятся справа от данной подзадачи.

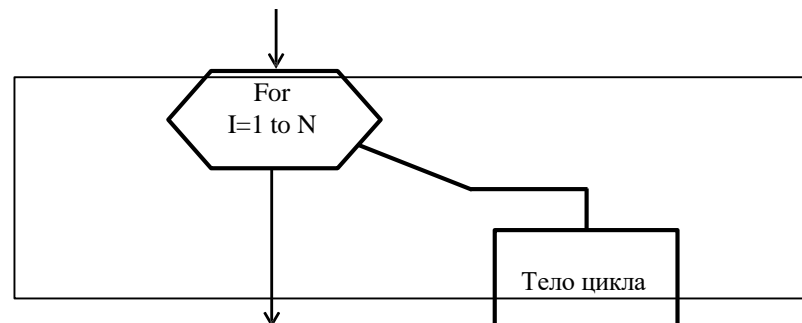


Рисунок 3.23 – Представление конструкции цикла с параметром по методу Дамке

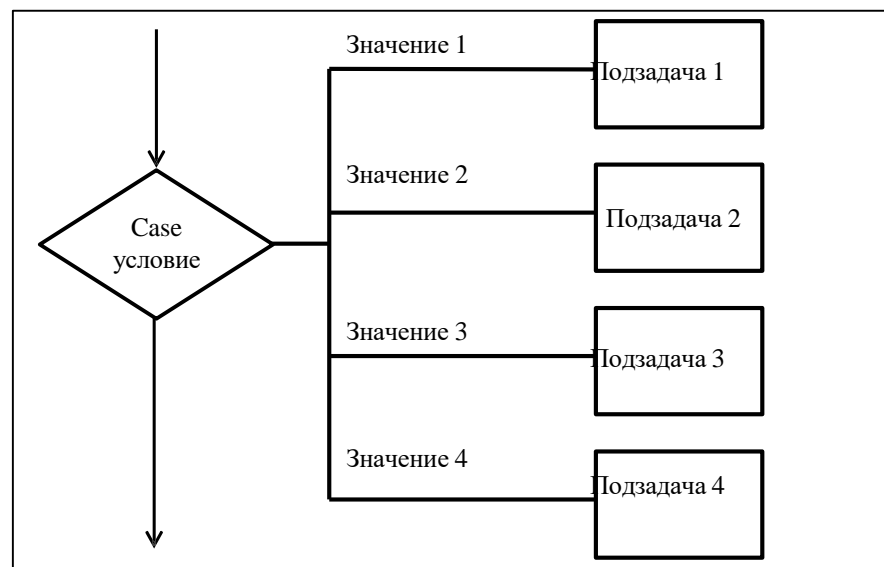


Рисунок 3.24 – Представление конструкции Case по методу Дамке

*Достоинства метода Дамке:*

- схема алгоритма, представленная с помощью данного метода, нагляднее, чем классическая, особенно для больших программ;
- метод Дамке удобно использовать при разработке алгоритма по методу нисходящего проектирования;
- метод Дамке удобен при коллективной разработке программ, так как позволяет независимо разрабатывать отдельные подзадачи.

## **Схемы Насси-Шнейдермана**

*Схемы Насси-Шнейдермана* – это схемы, иллюстрирующие структуру передач управления внутри модуля с помощью вложенных друг в друга блоков. Схемы используются для изображения структурированных схем и позволяют уменьшить громоздкость схем за счёт отсутствия явного указания линий перехода по управлению.

Схемы Насси-Шнейдермана называют ещё **структурограммами**.

Изображение основных элементов структурного программирования в схемах Насси-Шнейдермана организовано следующим образом. Каждый блок имеет форму прямоугольника и может быть вписан в любой внутренний прямоугольник любого другого блока. Информация в блоках записывается по тем же правилам, что и в структурированных схемах алгоритмов (на естественном языке или языке математических формул).

Изображение конструкций структурированных алгоритмов в схемах Насси-Шнейдермана описано ниже.

**1) Функциональный блок (блок обработки)** изображается так, как иллюстрирует рисунок 3.26.

Каждый символ схем Насси-Шнейдермана является блоком обработки.

Каждый прямоугольник внутри любого символа также является блоком обработки.

**2) Блок следования** изображается так, как представляет рисунок 3.27.

Данный блок объединяет ряд следующих друг за другом процессов обработки.

**3) Блок решения** изображается так, как показывает рисунок 3.28.

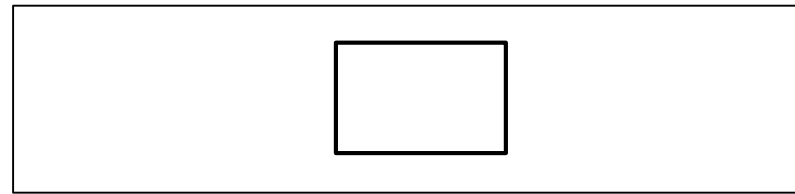


Рисунок – Представление функционального блока в схемах Насси-Шнейдермана

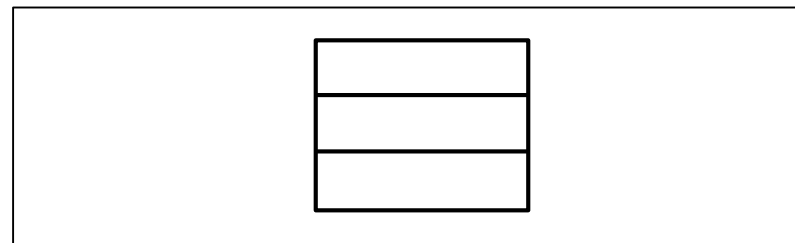


Рисунок – Представление блока следования в схемах Насси-Шнейдермана

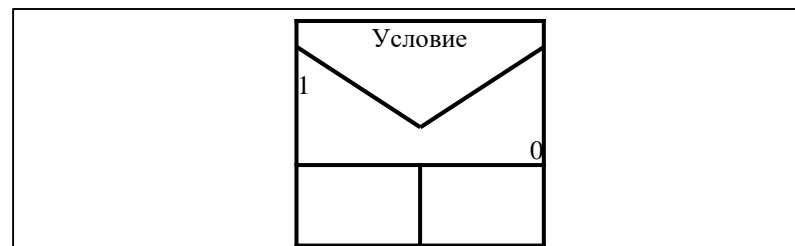


Рисунок 3.28 – Представление блока решения в схемах Насси-Шнейдермана

Блок решения используется для представления конструкции If-Then-Else. Условие записывается в центральном треугольнике, варианты исполнения условия – в боковых треугольниках (варианты исполнения могут быть записаны в виде: *1, 0; да, нет; +, -*). Процессы обработки обозначаются прямоугольниками.

4) **Блок Case** изображается так, как представляет рисунок 3.29.

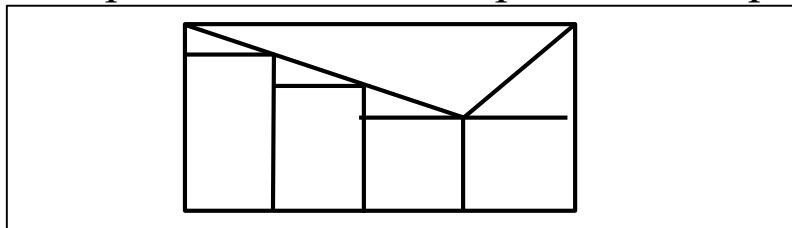


Рисунок – Представление блока Case  
в схемах Насси-Шнейдермана

Данный блок является расширением блока решения. Те варианты выхода из этого блока, которые можно точно сформулировать, размещаются слева от нижней вершины центрального треугольника. Остальные выходы объединяются в один, называемый выходом по несоблюдению условий и расположенный справа от нижней вершины.

Если можно перечислить все возможные случаи, правую часть можно оставить незаполненной или совсем опустить, а выходы разместить по обе стороны центрального треугольника.

По аналогии с конструкцией If-Then-Else условие записывается в центральном треугольнике, варианты исполнения условия – в боковых треугольниках. Процессы обработки обозначаются прямоугольниками.

5) **Цикл “Пока”** изображается так, как иллюстрирует рисунок 3.30.

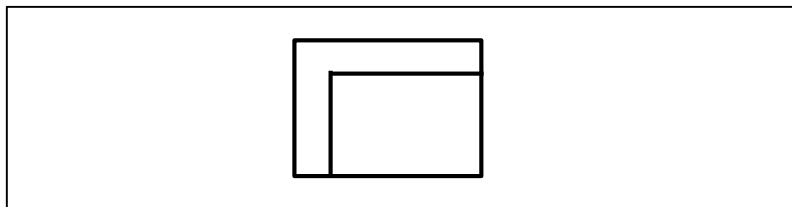


Рисунок – Представление цикла “Пока”  
в схемах Насси-Шнейдермана

Обозначает циклическую конструкцию с предусловием, т.е. с проверкой условия в начале цикла (цикл While). Условие выполнения цикла размещается в верхней полосе.

6) Цикл “До” изображается так, как представляет рисунок 3.31.

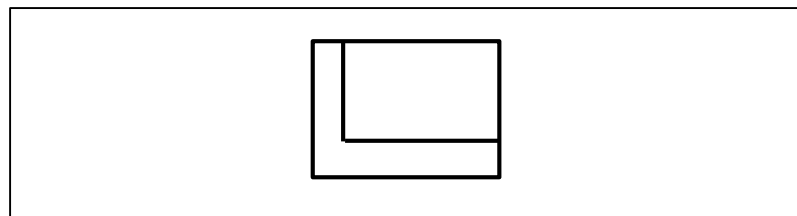


Рисунок – Представление цикла “До”  
в схемах Насси-Шнейдермана

Обозначает циклическую конструкцию с постусловием, т.е. с проверкой условия после выполнения тела цикла (цикл Repeat-Until). Условие выполнения цикла размещается в нижней полосе.

Рисунок 3.32 – рисунок 3.34 представляют примеры простейших структурированных схем алгоритмов (слева) и соответствующие им схемы Насси-Шнейдермана (справа).

На данных рисунках ***Bi*** обозначает *i*-ое условие, ***Di*** – *i*-ое действие.

Из данных рисунков видно, что схемы Насси-Шнейдермана являются существенно более компактными по сравнению с аналогичными схемами алгоритмов, представленными в соответствии с требованиями *ГОСТ 19.701-90* (см. п. 2.2.2). Очевидно, что связано это с отсутствием линий, отображающих потоки управления (линии перехода по управлению) между блоками.

#### Пример

Дан массив *A*, состоящий из *N* элементов. Найти наибольший из элементов массива (***Amax***) и его номер (***Imax***). Нарисовать укрупненную и детализированную схемы Насси-Шнейдермана.

Схемы Насси-Шнейдермана для решения данной задачи содержит рисунок 3.35.

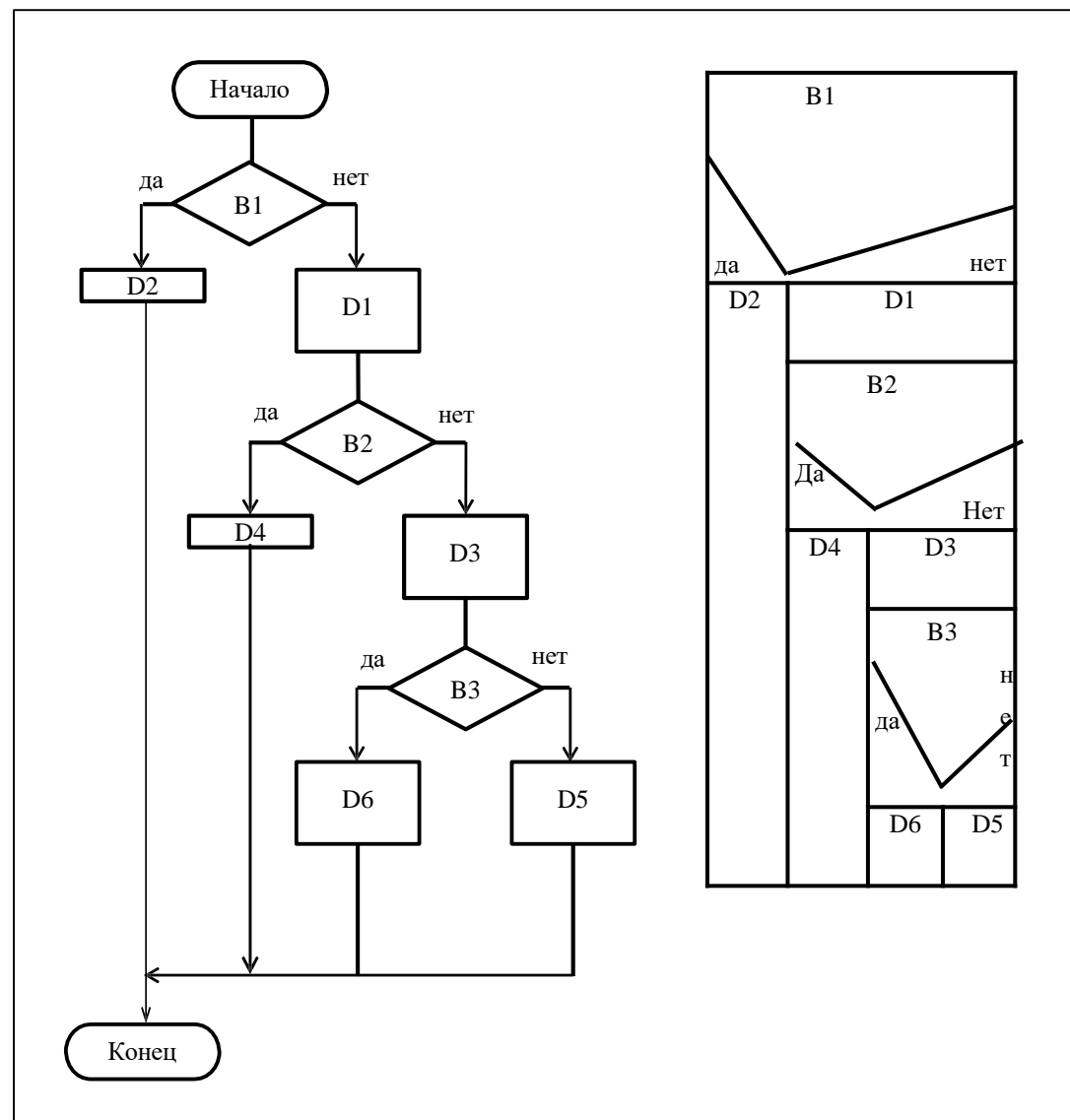
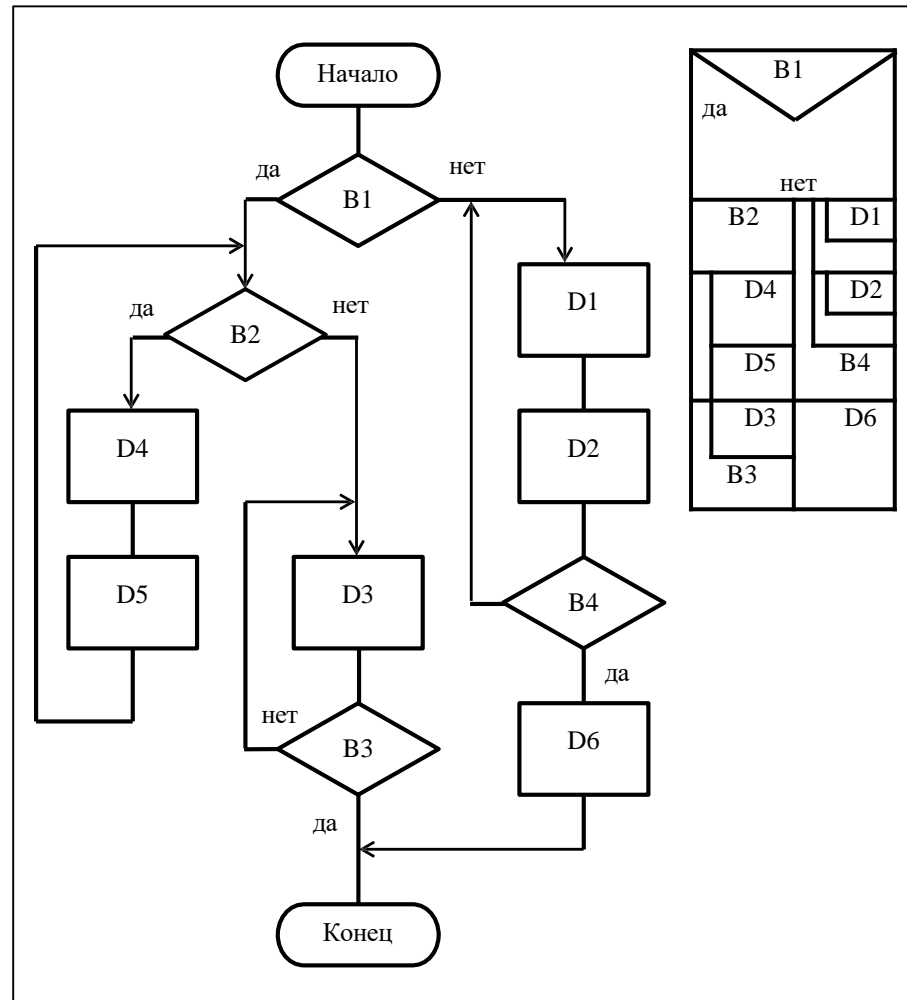
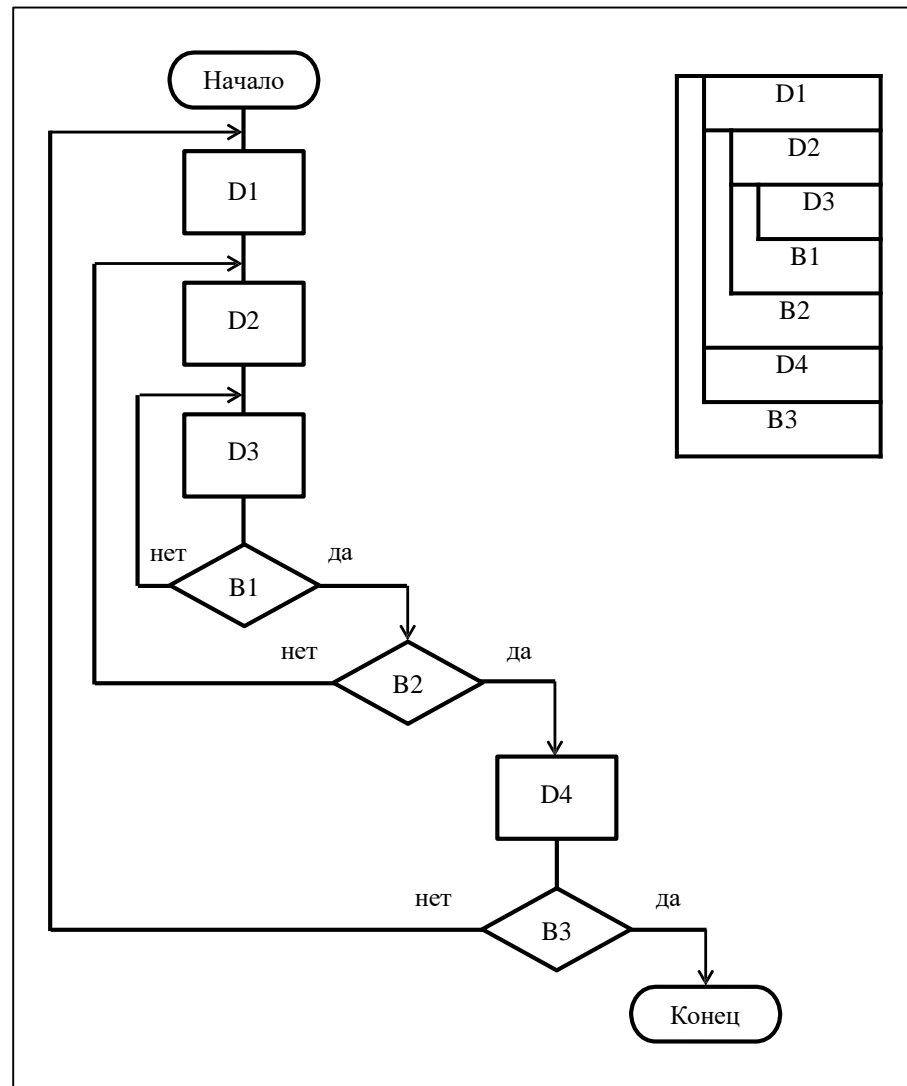


Рисунок – Пример схемы Насси-Шнейдермана для алгоритма с вложенными разветвлениями

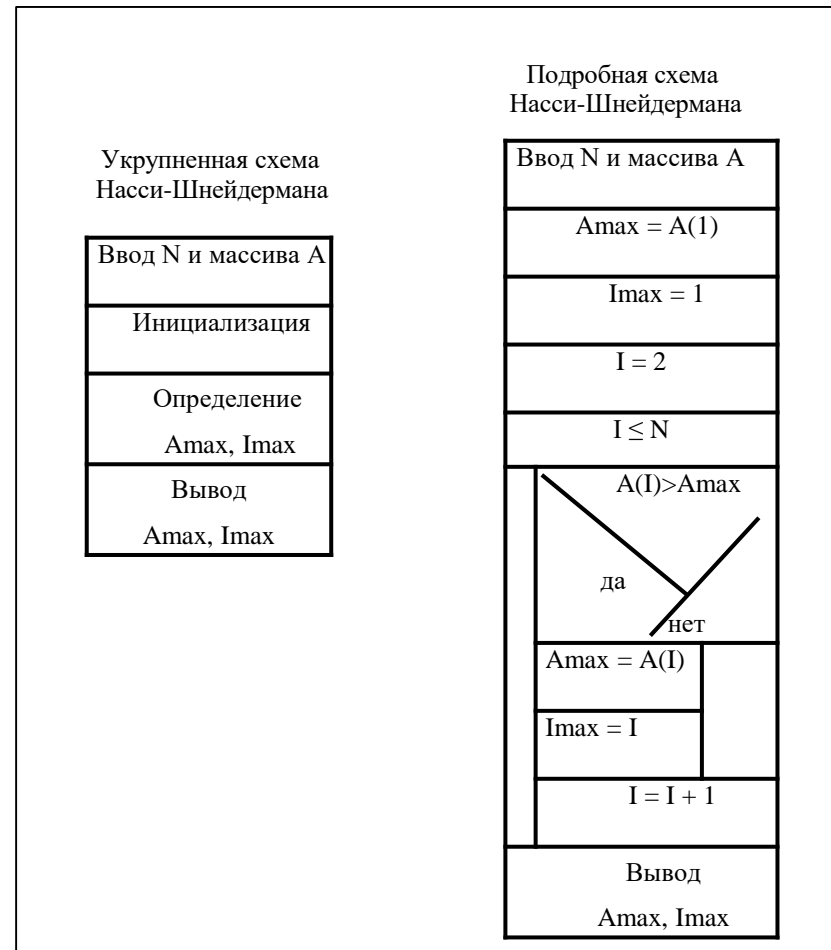




Рисунок– Пример схемы Насси-Шнейдермана для алгоритма, содержащего последовательность циклов



Рисунок– Пример схемы Насси-Шнейдермана для алгоритма, содержащего вложенные циклы



Рисунок— Пример диаграммы Насси-Шнейдермана  
для алгоритма поиска максимального элемента массива и его номера

## **Задание на лабораторную работу:**

**1. Представить индивидуальный программный проект (ВКР) в виде алгоритмов и в виде схем Несси-Шнейдермана.**

**2. Ответьте письменно (в отчете по лабораторной работе) на следующие вопросы:**

- a) Что такое алгоритм?
- b) Перечислите основные особенности алгоритмов.
- c) Назовите способы представления алгоритмов.
- d) Что такое блок-схема, каковы правила составления блок-схем?
- e) Какой алгоритм называется линейным?
- f) Перечислите виды разветвляющихся алгоритмов.
- g) Какие виды циклов вы знаете?
- h) Объясните работу циклов со счетчиком.
- i) В чем отличие циклов с предусловием от циклов с постусловием?
- j) Что такое сложные циклы?
- k) Для чего необходимы схемы Несси-Шнейдермана и в чем их достоинства?