Тема 1.

Алгоритмизация вычислительных процессов

Содержание

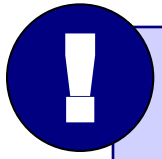
1. Основные определения и понятия
2. Этапы разработки программ
3. Методы проектирования алгоритмов и программ
4. Принципы структурного программирования
5. Понятия алгоритма и их свойства
6. Способы отображения алгоритмов
7. Базовые алгоритмические структуры

1. Проектирование программ

Этапы разработки программ

1. Постановка задачи

- определить **цель** и **категорию** программы (системная, прикладная)
- определить **исходные данные** и требуемый **результат**
- проверить, является ли задача **хорошо поставленной** (должны быть определены все связи между исходными данными и результатом)



Плохо поставленные задачи:

- не хватает исходных данных
- заданы не все связи между исходными данными и результатом
- задача не имеет решения
- задача имеет множество решений

- Зафиксировать требования к программе **в письменной форме**

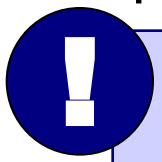
Этапы разработки программ

2. **Разработка модели данных**
   - Анализ существующих аналогов
   - формальная модель
   - типы данных (массивы, структуры, …)
   - взаимосвязь между данными
3. **Разработка алгоритма**
   - выбор существующего или разработка нового
   - возможен возврат к шагу 2
4. **Разработка программы**

   Языки: C, C++, Visual Basic, Паскаль, …

Этапы разработки программ

5. **Отладка программы** (поиск и исправление ошибок)
   - **отладчик** (точки останова, пошаговый режим, просмотр переменных)
   - **профайлер** (сколько выполняется каждая из процедур)
6. **Тестирование программы** (проверка на исходных данных, для которых известен результат)
   - **альфа**-тестирование: внутри фирмы (тестеры)
   - **бета**-тестирование: в других организациях, распространение через Интернет



Тестирование может показать наличие ошибок, но не их отсутствие.

Этапы разработки программ

***Отладка** программы* – это процесс поиска и устранения ошибок в программе, производимый по результатам её прогона на компьютере.

Тестирование - это составление специальных наборов входных и выходных данных (тестов), а затем исполнение программы и проверка полученных результатов в поисках возможных семантических или логических ошибок.

Этапы разработки программ

Отладка и тестирование – это два четко различимых и непохожих друг на друга этапа:

- при **отладке** происходит локализация и устранение синтаксических ошибок и явных ошибок кодирования;
- в процессе же **тестирования** проверяется работоспособность программы, не содержащей явных ошибок.

Тестирование устанавливает факт наличия ошибок, а отладка выясняет ее причину.

Этапы разработки программ

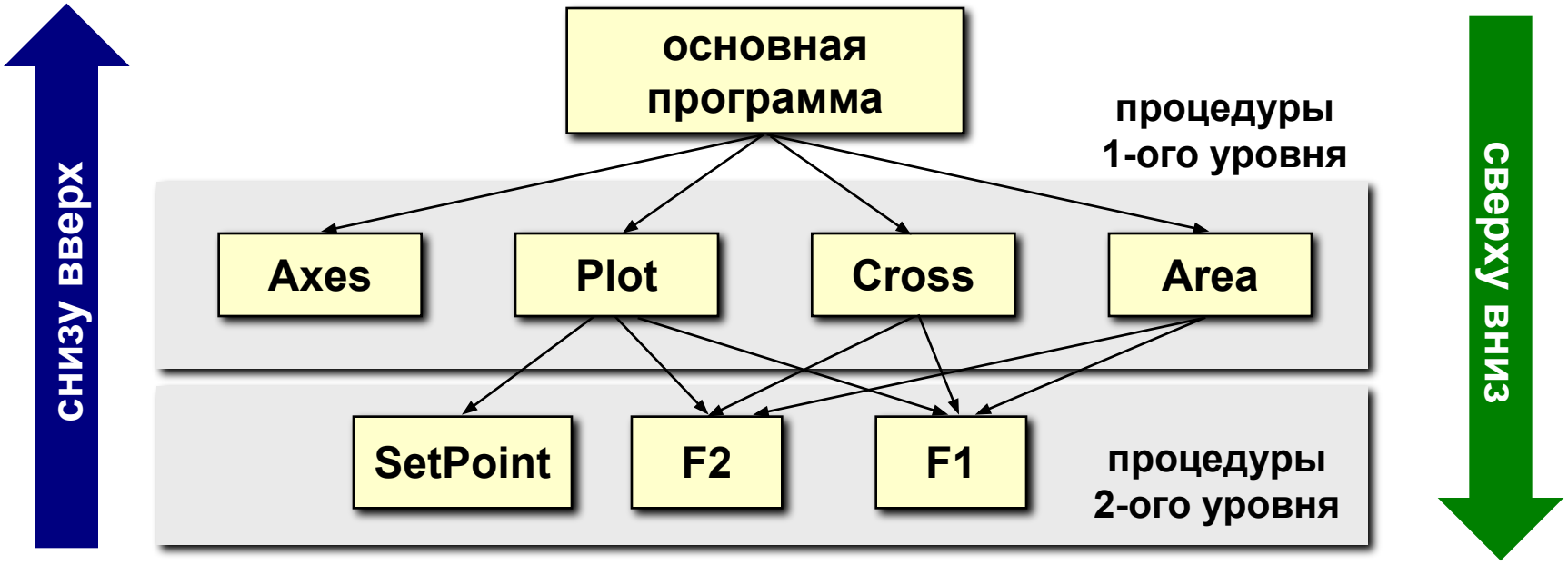
Английский термин *debugging* ("*отладка*") буквально означает "*вылавливание жучков*". Термин появился в 1945г., когда один из первых компьютеров – "Марк–1" прекратил работу из–за того, что в его электрические цепи попал мотылек и заблокировал своими останками одно из тысяч реле машины.

В современных программных системах отладка осуществляется с использованием специальных программных средств, называемых ***отладчиками***. Эти средства позволяют исследовать внутреннее поведение программы.

Этапы разработки программ

7. **Разработка документации**
   - справочная система
   - руководство пользователя (*User Manual*)
   - руководство разработчика
8. **Сопровождение (техническая поддержка)**
   - исправление ошибок, найденных заказчиком
   - обучение и консультирование заказчика
   - новые версии по льготной цене

Методы проектирования программ



Проектирование «снизу вверх»

сначала составляются процедуры нижнего уровня, из которых затем «собираются» процедуры более высокого уровня.

- **легче начать** программировать
- более **эффективные** процедуры

- процедуры необходимо связывать с основной задачей («**держать в голове**»)
- при окончательной сборке может **не хватить** «кубиков»
- часто программа получается **запутанной**
- сложно распределить работу **в команде**

Проектирование «сверху вниз»

метод последовательного уточнения:

1) начинаем **с основной программы**;
2) она разбивается на подзадачи, для каждой из которых пишется процедура;
3) реализуем каждую из процедур тем же способом.

- меньше вероятность **принципиальной ошибки** (начали с главного)
- проще **структура** программы
- удобно **распределять** работу в команде

- в разных блоках могут быть реализованы похожие операции (можно было решить одной **общей процедурой**), особенно в команде

Структурное программирование

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков

Существовавшие проблемы:

- увеличилась **сложность** программ
- сократилось **время** на разработку

Цели:

- повысить **надежность**
- уменьшить **время** и **стоимость** разработки
- облегчить тестирование и **отладку**
- возможность **переделки** одного модуля
- улучшить **читабельность**

Структурное программирование

Принципы:

- **абстракции**: программу можно рассматривать на любом уровне без лишних подробностей
- **модульности**: программа разбивается на отдельные модули, которые могут отлаживаться независимо друг от друга
- **подчиненности**: связь между модулями «сверху вниз»
- **локальности**: каждый модуль использует только свои локальные переменные, глобальные переменные только в крайних случаях

Модуль

Модуль – это программный блок (процедура или функция), отделенный от кода других модулей, который полностью решает самостоятельную задачу своего уровня.

- работа модуля не зависит от того, **откуда** он вызывается, и от того, сколько раз он вызывался **до этого**
- размер модуля не более **50-60 строк** (1 страница)
- модуль имеет **один вход** и один **выход**
- модуль начинается с «шапки»-**комментария** (входные данные, результаты, какие модули использует)
- имена переменных – **смысловые**
- в одной строке – один оператор

Модуль

```
Program Max;
var x, y, m, n: integer;

procedure MaxNumber(a,b: integer; var max: integer);
begin
   if a>b then max:=a else max:=b;
end;

begin
   write('Введите x, y ');
   readln(x, y);
   MaxNumber(x, y, m);
   MaxNumber(2, x+y, n);
   writeln('m=', m, 'n=', n);
end.
```

Основные определения и понятия

Алгоритмизация – это процесс построения алгоритма решения задачи, результатом которого является выделение этапов процесса обработки данных, формальная запись содержания этих этапов и определение порядка их выполнения.

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

Основные определения и понятия

Свойства алгоритма

- **дискретность (раздельность)**: состоит из отдельных шагов (команд); возможность расчленения вычислительного процесса на отдельные элементарные операции, возможность выполнения которых не вызывает сомнений
- **понятность**: должен включать только команды, известные исполнителю
- **определенность**: при одинаковых исходных данных всегда выдает один и тот же результат; точность указаний, исключающая их произвольное толкование
- **конечность**: заканчивается за конечное число шагов

Основные определения и понятия

Свойства алгоритма

- **массовость**: может применяться многократно при различных исходных данных; пригодность алгоритма для решения всех задач заданного класса
- **корректность**: дает верное решение при любых допустимых исходных данных

Основные определения и понятия

Программа – это

- алгоритм, записанный на каком-либо языке программирования
- набор команд для компьютера

Команда – это описание действий, которые должен выполнить компьютер.

- откуда взять исходные данные?
- что нужно с ними сделать?

Основные определения и понятия

Программа – алгоритм, записанный в форме, воспринимаемой машиной.

Программа содержит наряду с описанием данных команды, в какой последовательности, над какими ***данными*** и какие операции должна выполнять машина, а также в какой форме следует получить результат.

Это обеспечивают различные ***операторы***.

Основные определения и понятия

Переменная – это объект, который в ходе выполнения программы может менять свое значение.

Свойства переменной:

1) переменная называется неопределенной до тех пор, пока она не получит значение:
   а) вводом извне;
   б) занесением константы;
   в) занесением значения другой, ранее определенной переменной;
2) в каждый момент времени переменная может либо иметь определенное значение, либо быть неопределенной;
3) последующее значение уничтожает (стирает) предыдущее значение. Выбор (чтение) переменной и ее использование не изменяют значение переменной.

Основные определения и понятия

Данные – это факты и идеи, представленные в формализованном виде, позволяющем передавать или обрабатывать эти факты и идеи с помощью некоторого процесса.

Оператор – совокупность символов, указывающих операцию и значения, либо местонахождение ее элементов.

```
A:= B+C; {A, B, C – переменные;}
K:= 2; IF T< 0 THEN . . .
```

Основные определения и понятия

Предметом курса являются методы и средства составления алгоритмов и программ с целью решения задач на ЭВМ. Для разработки программ используются *системы программирования*.

Система программирования – средство автоматизации программирования, включающее язык программирования, транслятор этого языка, документацию, а также средства подготовки и выполнения программ.

Системы программирования

Системы программирования (или инструментальные средства) – это ПО, предназначенное для разработки и отладки новых программ.

Проблема:

- компьютеры понимают только **язык кодов** (последовательность нулей и единиц)
- для человека удобнее давать задания на **естественном языке** (русском, английском)

Компромисс:

программы составляются на **языках программирования** и затем переводятся в коды с помощью специальных программ

Языки программирования

Всего более 600, широко используется примерно 20.

Машинно-ориентированные языки:

- **машинные коды:** 09 FE AC 3F
- **ассемблеры:** символическая запись машинных команд:

  mov AX, BX
- **макроассемблеры:** одна команда языка заменяет несколько машинных команд

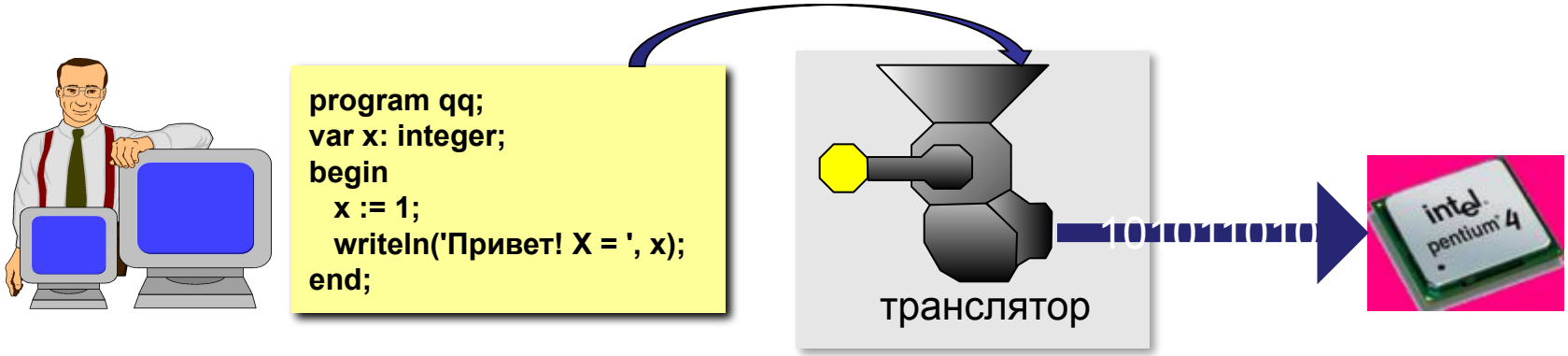
Языки программирования

Языки высокого уровня (алгоритмические):

- **для обучения**: Бейсик (1965), Паскаль (1970), Лого, Рапира
- **профессиональные:** Си (1972), Паскаль (Delphi), Фортран (1957), Visual Basic
- **для задач искусственного интеллекта**: ЛИСП, Пролог
- **для параллельных вычислений**: Ада
- **для программирования в Интернете**: JavaScript, Java, PHP, Perl, ASP, …

Трансляторы

Транслятор – это программа, которая переводит текст других программ в машинные коды.



программа на языке Паскаль

программа в машинных кодах

Типы трансляторов

- **интерпретатор** – переводит в коды 1 строчку программы и сразу ее выполняет;

  - удобнее отлаживать программу

  - программы работают медленно (цикл из 400 шагов!)
  - для выполнения программы нужен транслятор

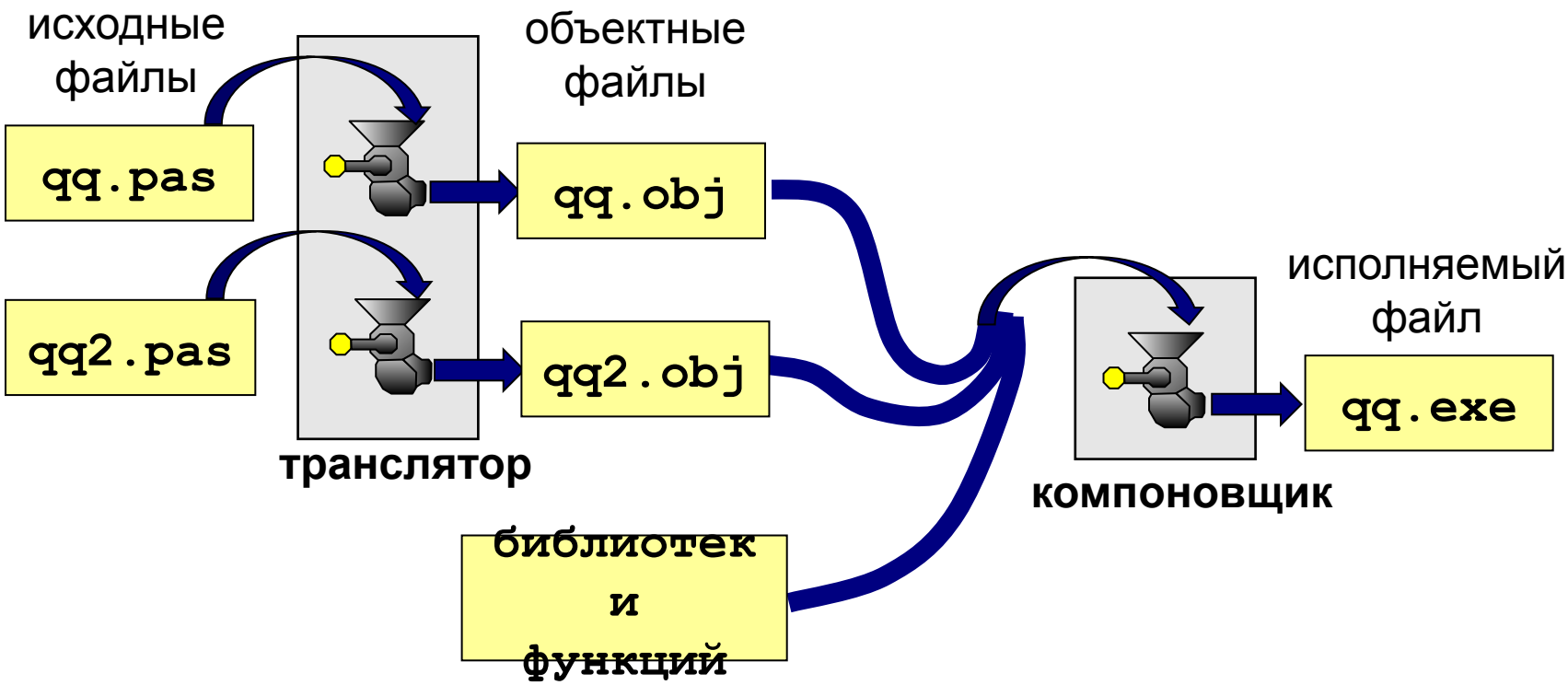
- **компилятор** – переводит в коды сразу всю программу и создает независимый исполняемый файл (***.exe**);

  - сложнее отлаживать программу

  - программы работают быстро
  - для выполнения программы не нужен транслятор

Компоновщик

Компоновщик (редактор связей, *Linker*) – это программа, которая объединяет части одной программы и библиотечные функции в один исполняемый файл.



Другие программы

Отладчик (англ. *debugger*) – это программа, которая облегчает поиск ошибок в других программах (их отладку).

Возможности:

- пошаговое выполнение
- «выполнить до курсора»
- просмотр и изменение значений переменных
- точки останова (англ. *breakpoints*)

Профайлер (англ. *profiler*) – это программа, которая определяет, сколько времени занимает выполнение каждой процедуры (и каждой команды) в программе в процентах от общего времени работы.

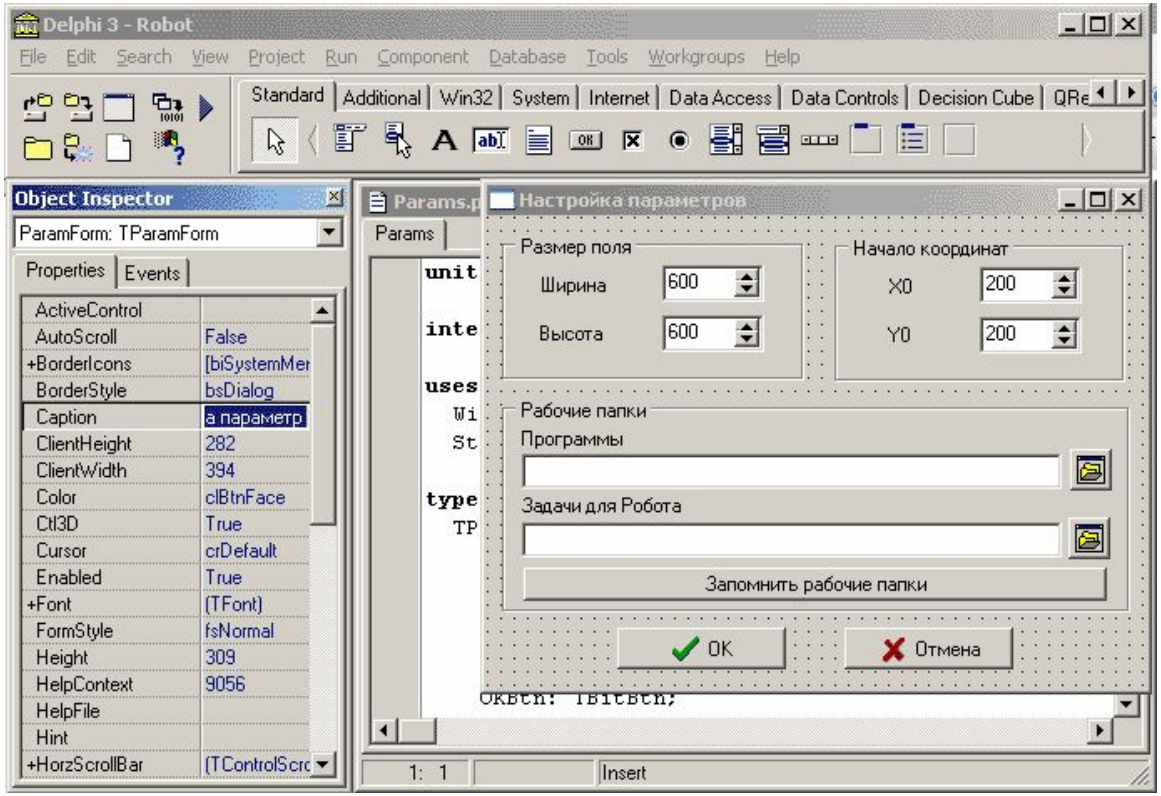
Цель: определить, какие части программы «тормозят» ее (англ. *bottleneck* – бутылочное горлышко), именно их и надо оптимизировать.

Среда быстрой разработки

Среда быстрой разработки программ (англ. *RAD = Rapid Application Development*)

- интерфейс строится с помощью мыши
- часть кода создается автоматически

Примеры: *Delphi, Borland C++ Builder, Visual Studio…*



Основные определения и понятия

Предметом курса являются методы и средства составления алгоритмов и программ с целью решения задач на ЭВМ. Для разработки программ используются *системы программирования*.

Система программирования – средство автоматизации программирования, включающее язык программирования, транслятор этого языка, документацию, а также средства подготовки и выполнения программ.

Транслятор – это программа, которая переводит с одного языка на другой.

Интерпретатор – это программа, которая сразу выполняет переводимые команды.

Компилятор – это программа, которая переводит конструкции алгоритмического языка в машинные коды.

2. Средства изображения алгоритмов

Средства изображения алгоритмов

Основными изобразительными средствами алгоритмов являются следующие способы их записи:

- словесный;
- формульно-словесный;
- блок-схемный;
- псевдокод;
- структурные диаграммы;
- языки программирования.

Словесный способ

Словесный – содержание этапов вычислений задается на естественном языке в произвольной форме с требуемой детализацией.

1) отсутствует наглядность вычислительного процесса, т.к. нет достаточной формализации;
2) страдают многословностью записей;
3) допускают неоднозначность толкования отдельных предписаний.

Словесный способ

Пример 1. Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел (***алгоритм Эвклида***).

Алгоритм может быть следующим:

1. задать два числа;
2. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3. определить большее из чисел;
4. заменить большее из чисел разностью большего и меньшего из чисел;
5. повторить алгоритм с шага 2.

Словесный способ

Пример 2. Пусть задан массив чисел. Требуется проверить, все ли числа принадлежат заданному интервалу. Интервал задается границами A и B.

п.1 Берем первое число.
п.2 Сравниваем: выбранное число принадлежит интервалу; если да, то на **п.3**, если нет – на **п.6**.
п.3 Все элементы массива просмотрены? Если да, то на **п.5**, если нет – то на **п.4**.
п.4 Выбираем следующий элемент. На **п.2**.
п.5 Печать сообщения: все элементы принадлежат интервалу. На **п.7**.
п.6 Печать сообщения: не все элементы принадлежат интервалу. На п.7.
п.7 Конец.

Формульно-словесный способ

Формульно-словесный – задание инструкций с использованием математических символов и выражений в сочетании со словесными пояснениями.

Пример. Требуется написать алгоритм вычисления площади треугольника по трем сторонам.

п.1 – вычислить полупериметр треугольника $p=(a+b+c)/2$.

п.2 – вычислить $S = \sqrt{p(p-a)(p-b)(p-c)}$

п.3 – вывести S , как искомый результат и прекратить вычисления.

При использовании этого способа может быть достигнута любая степень детализации, более наглядно, но не строго формально.

Блок-схемный способ

Блок-схемный – это графическое изображение логической структуры алгоритма, в котором каждый этап процесса переработки данных представляется в виде геометрических фигур (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций.

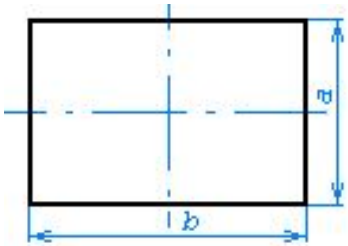
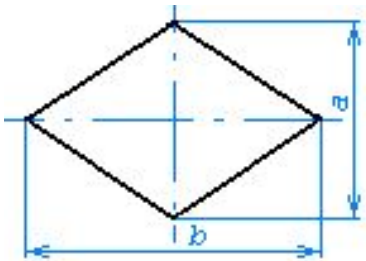
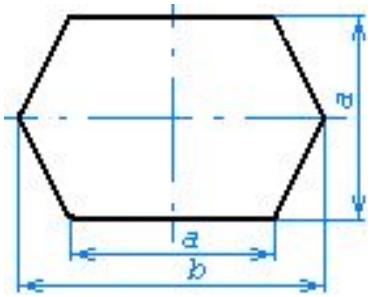
Блочные символы соединяются *линиями переходов*, определяющими очередность выполнения действий.

Далее приведены наиболее часто употребляемые символы (ГОСТ 19002-80 и ГОСТ 19003-80 "Схемы алгоритмов и программ").

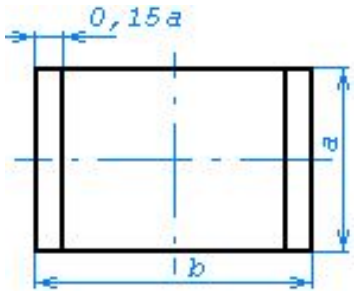
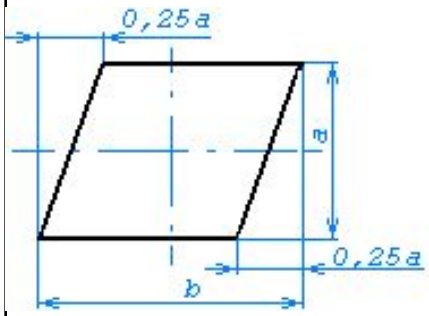
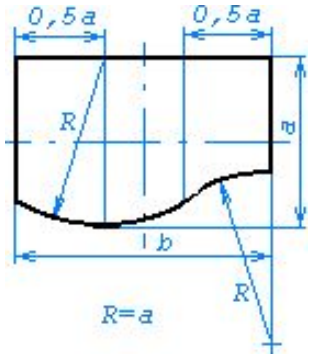
Соотношение геометрических размеров символов

Размер a должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер a на число, кратное 5. Размер b равен 1,5a.

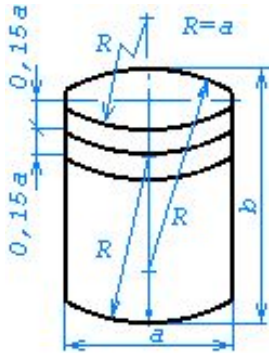
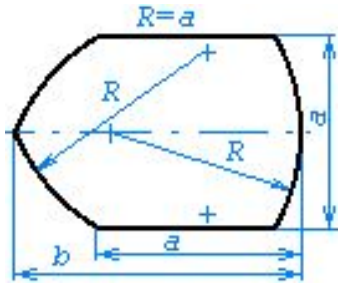
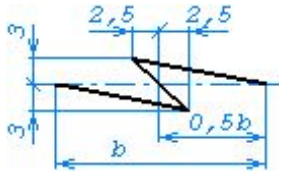
Основные символы блок-схем

Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных
Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
Модификация		Выполнение операций, меняющих команды, или группы команд, изменяющих программу (организация цикла)

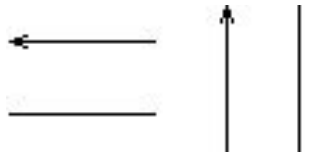
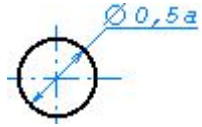
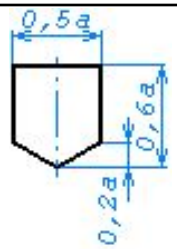
Основные символы блок-схем

Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов или программ
Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод)
Документ		Ввод-вывод данных, носителем которых служит бумага

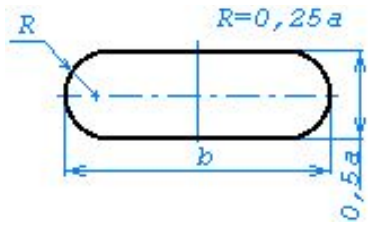
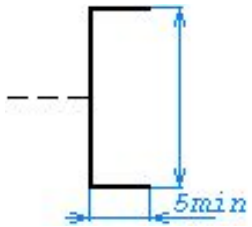
Основные символы блок-схем

Магнитный диск		Ввод-вывод данных, носителем которых служит магнитный диск
Дисплей		Ввод-вывод данных, если непосредственно подключенное к процессору устройство воспроизводит данные и позволяет оператору ЭВМ вносить изменения в процесс их обработки
Канал связи		Передача данных по каналам связи

Основные символы блок-схем

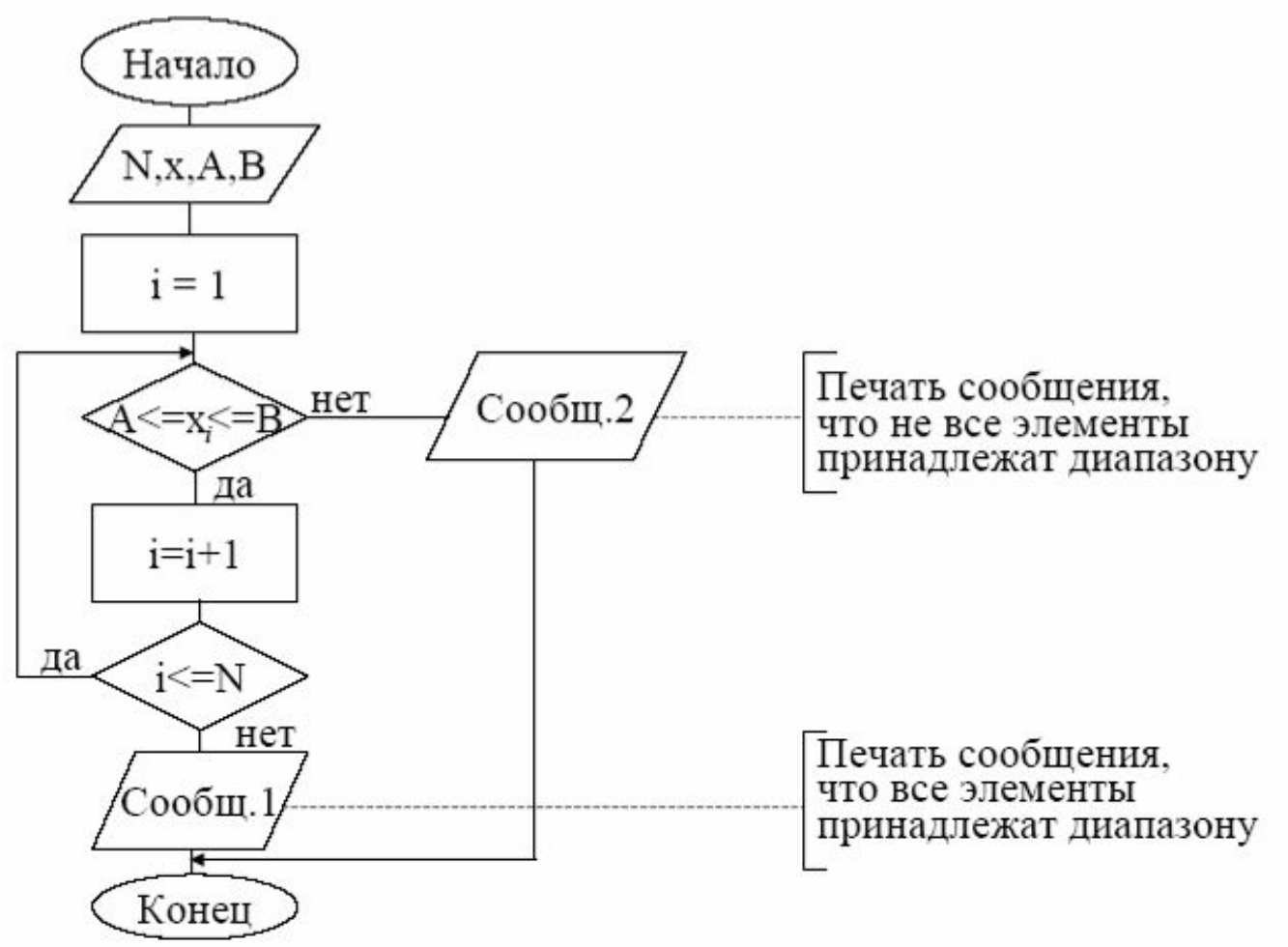
Линия потока		Указание последовательности связей между символами
Соединитель		Указание связи между прерванными линиями потока, соединяющими символы
Межстранич-ный соедини-тель		Указание связи между разъединенными частями схем алгоритмов и программ, расположенных на разных листах

Основные символы блок-схем

Пуск-останов		Начало, конец, прерывание процесса обработки данных или выполнения программы
Комментарий		Связь между элементом схемы и пояснением

Пример блок-схемы

Рассмотрим пример блок-схемы той же задачи, для которой приведен словесный алгоритм.



Псевдокод

Псевдокод - позволяет формально изображать логику программы, не заботясь при этом о синтаксических особенностях конкретного языка программирования.

Обычно представляет собой смесь операторов языка программирования и естественного языка.

Является средством представления логики программы, которое можно применять вместо блок-схемы.

Псевдокод

Шаги алгоритма и последовательность их выполнения задаются с помощью набора специальных ***ключевых слов***.

Существует много различных вариантов псевдокода, например, в русскоязычной литературе распространен следующий вариант псевдокода:

- **нач** -- начало программы;
- **кон** -- конец программы;
- **если ... то ...иначе** -- проверка условия;
- **ввод** -- ввод данных;
- **вывод** -- вывод данных;
- **для ... от .. до ... нц ... кц** -- цикл со счетчиком (нц -- начало цикла, кц -- конец);
- **пока ... нц ...кц** -- цикл с предусловием;
- **нц ... кц ... пока** -- цикл с постусловием.

Пример псевдокода

Выбираем первый элемент (i=1)
IF A>x_i или x_i<B THEN
печать сообщения и переход на конец
ELSE переход к следующему элементу(i=i+1)
IF массив не кончился (i<=n) THEN
переход на проверку интервала
ELSE печать сообщения, что все элементы входят в интервал
Конец

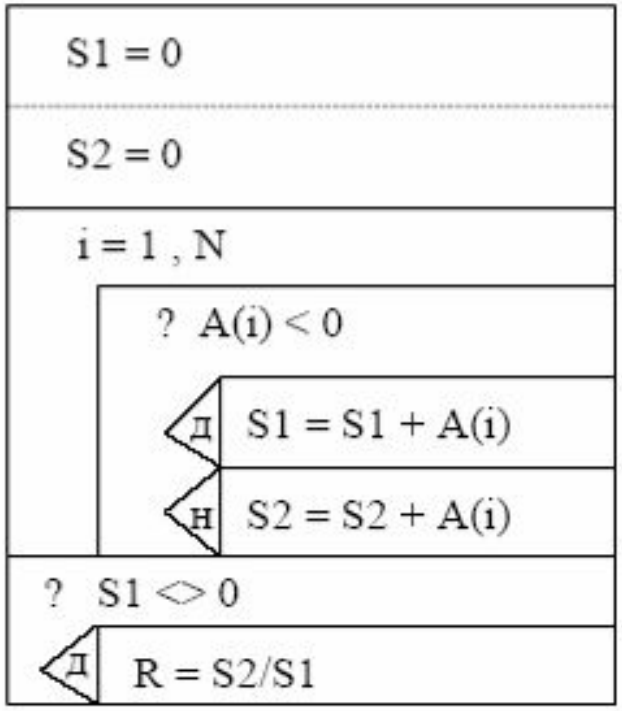
Структурные диаграммы

Структурные диаграммы - могут использоваться в качестве структурных блок-схем, для показа межмодульных связей, для отображения структур данных, программ и систем обработки данных.

Существуют различные структурные диаграммы: диаграммы Насси-Шнейдермана, диаграммы Варнье, Джексона, МЭСИД и др.

Структурные диаграммы

Пример диаграммы МЭСИД




```
. . . S1:=0;
     S2:=0;
FOR I :=1 TO N DO
     IF A[I] < 0 THEN
                S1:=S1 + A[I]
                   ELSE
                S2:=S2 +A[I] ;
IF S1 <> 0 THEN
                R:= S2/S1; . . .
```

Языки программирования

Языки программирования - изобразительные средства для непосредственной реализации программы на ЭВМ.

Каждая машина имеет свой собственный язык (машинный язык) и может выполнять программы только на этом языке. Это последовательность машинных команд.

Писать программы на машинном языке очень сложно и утомительно. Для повышения производительности труда программистов применяются искусственные языки программирования.

При этом требуется перевод программы, написанной на таком языке, на машинный язык. Этот перевод выполняет ***транслятор***.

Языки программирования

Наиболее часто встречающимся транслятором *интерпретирующего типа* является транслятор с языка Бейсик, где команды читаются, преобразуются и выполняются сразу. Итогом работы такого транслятора являются требуемые результаты.

Транслятор с Паскаля – *компилирующего типа*: сначала весь текст программы на языке Паскаль переводится в текст на машинном языке (получается, так называемый, ***объектный модуль*** программы), который затем обрабатывается Редактором межпрограммных связей и только после этого программа будет готова к выполнению.

Языки программирования

Любой компилятор требует, чтобы программа, подаваемая ему для перевода, была абсолютно правильно составлена.

В языке программирования, как и в любом другом языке, существуют ***синтаксис*** - правила записи его конструкций - и ***семантика*** - смысл его конструкций.

Компилятор проверяет только *синтаксис*.

Поиском же семантических ошибок занимается программист в процессе *тестирования* и *отладки* своей *программы.*

3. Базовые канонические структуры алгоритмов

Доказано, что любую программу можно написать, используя комбинации трех управляющих структур:

- следование (линейный алгоритм);
- ветвление (разветвляющийся алгоритм);
- цикл (циклический алгоритм).

Программа, составленная из канонических структур, будет называться ***регулярной программой***, т.е. иметь 1 вход и 1 выход, каждый оператор в программе может быть достигнут при входе через ее начало (нет недостижимых операторов и бесконечных циклов). Управление в такой программе передается сверху-вниз.

Снабженные комментариями, такие программы хорошо читабельны.

Линейные алгоритмы

Линейными называют алгоритмы, в которых операции выполняются последовательно одна за другой, в естественном и единственном порядке следования.

В таких алгоритмах все блоки имеют последовательное соединение логической связью передачи информационных потоков. В алгоритмах с линейной структурой каждый оператор выполняется только одни раз и выполняется весь целиком, т.е. выполняются все ее операторы.

В них могут использоваться все блоки, за исключением блоков проверки условия и модификации.

Линейные алгоритмы, как правило, являются составной частью любого алгоритмического процесса.

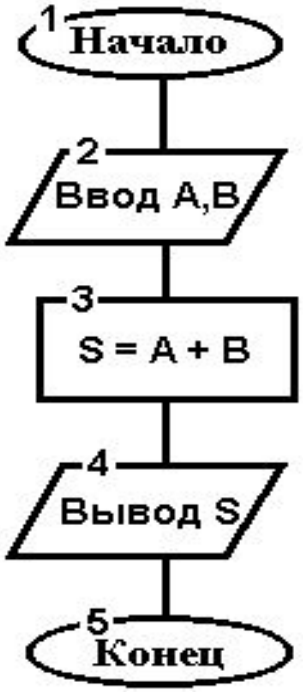
Линейные алгоритмы

Задача:

Даны A, B

Найти S=A+B

Схема алгоритма:



Разветвляющиеся алгоритмы

При составлении схем алгоритмов часто возникает необходимость проведения анализа исходных данных или промежуточных результатов вычислений и определения дальнейшего порядка выполнения вычислительного процесса в зависимости от результатов этого анализа.

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.

Разветвляющиеся алгоритмы

Каждое возможное направление вычислений называется *ветвью*.

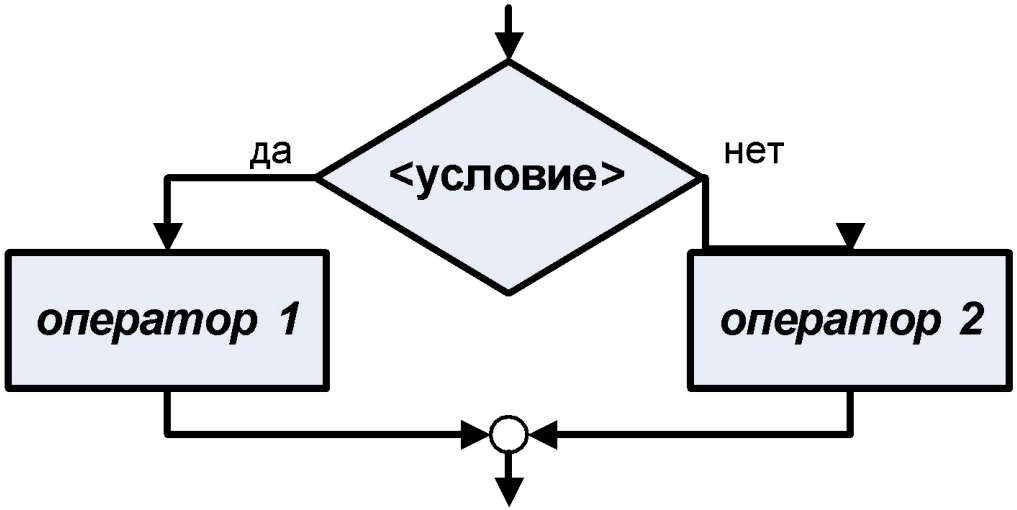
Каждая из ветвей ведет к *общему выходу*, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.

Структура ***ветвление*** существует в трех основных вариантах:

- если–то (неполная форма ветвления);
- если–то–иначе (полная форма ветвления);
- выбор.

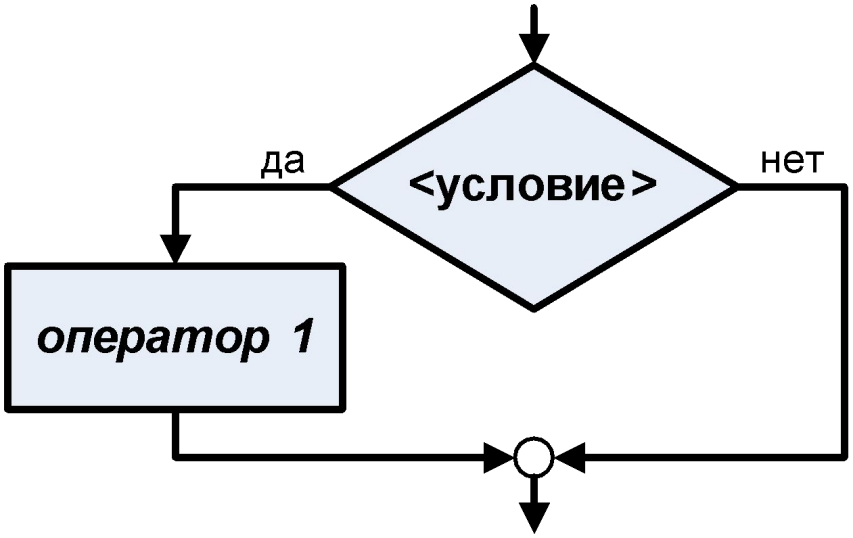
Разветвляющиеся алгоритмы

Полная форма ветвления



Разветвляющиеся алгоритмы

Неполная форма ветвления

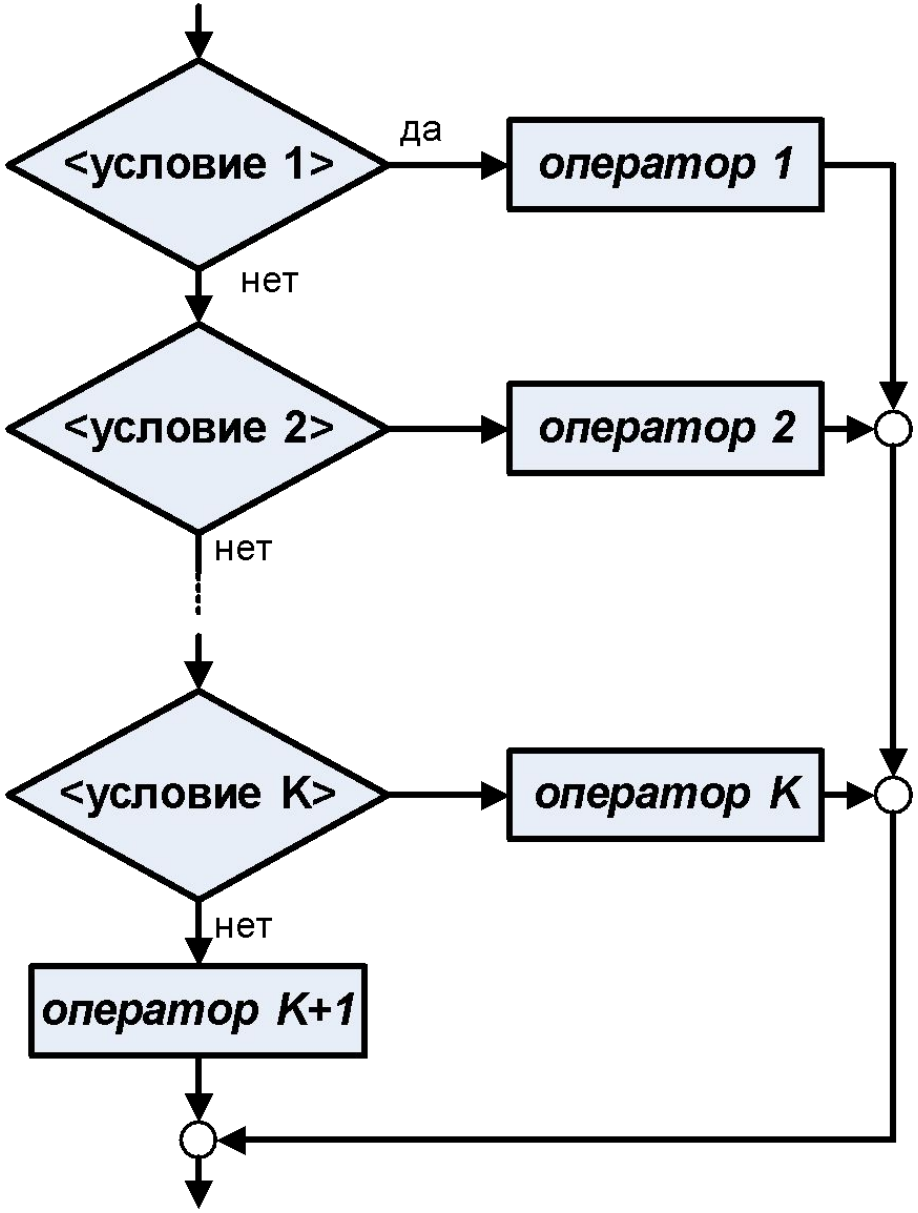


Разветвляющиеся алгоритмы

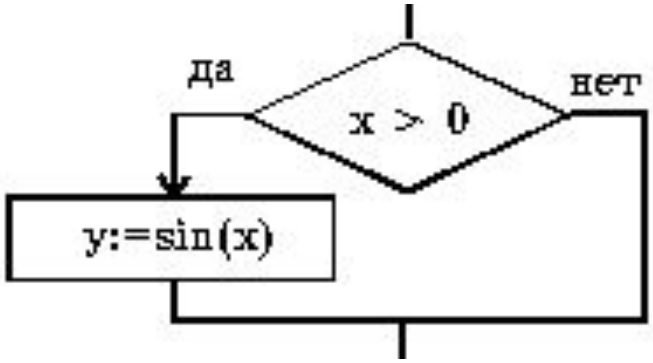
Выбор

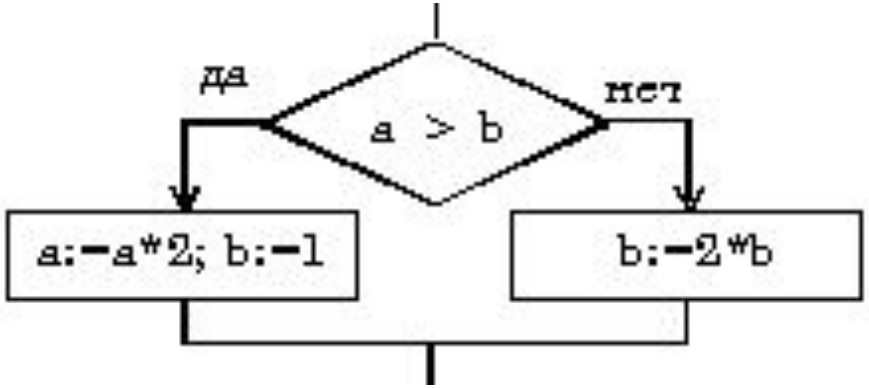
Оператор выбора варианта проверяет поочередно условия и выбирает для исполнения оператор, соответствующий условию, значение которого истинно.

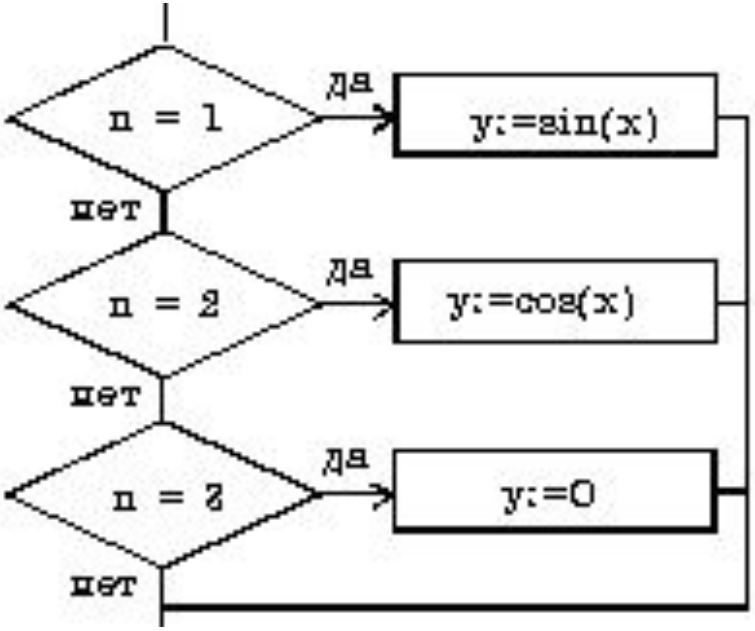
Если все ***K*** условий ложны, то выбирается оператор, стоящий по ветке ***нет*** последнего условия (последний в списке вариантов).

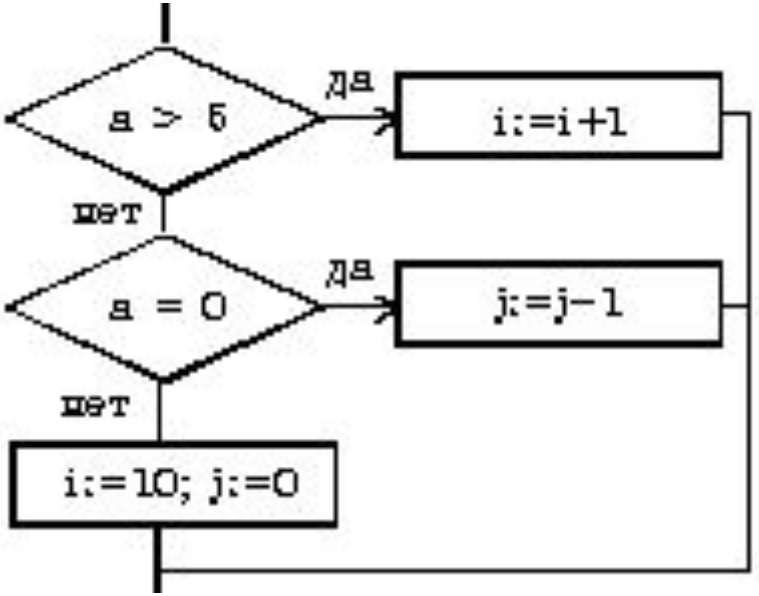


Разветвляющиеся алгоритмы





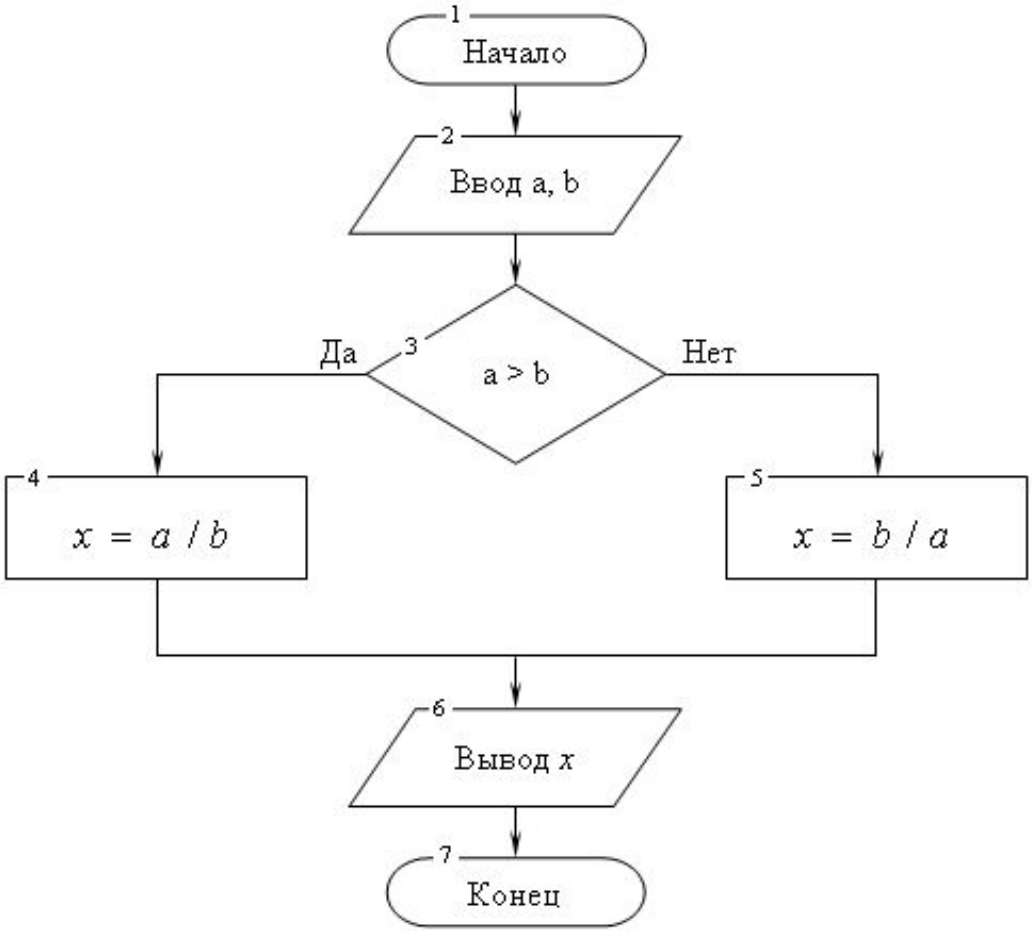




Разветвляющиеся алгоритмы

Даны два числа *a* и *b*. Найти

$$x = \begin{cases} a/b, \text{если } a > b \\ b/a, \text{если } a \le b \end{cases}$$



Циклические алгоритмы

Алгоритм ***циклической*** структуры предусматривает многократное повторение действий в одной и той же последовательности по одним и тем же математическим зависимостям, но при разных значениях некоторой специально изменяемой величины.

Циклические алгоритмы позволяют существенно сократить объем программы за счет многократного выполнения группы повторяющихся вычислений, так называемого ***тела цикла***.

Циклические алгоритмы

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 тому подобные величины.

Во время выполнения тела цикла параметры переменной цикла изменяются в интервале от начального до конечного значения с заданным шагом.

Циклические алгоритмы

Следовательно, при организации циклических вычислений необходимо предусмотреть:

- ✔ задание начального значения переменной цикла,
- ✔ закон ее изменения перед каждым новым повторением
- ✔ и ее конечное значение, при достижении которого произойдет завершение цикла.

Циклические алгоритмы

Циклы, в теле которых нет разветвлений и других встроенных в них циклов, называют ***простыми***. В противном случае их относят к ***сложным***.

Циклические алгоритмы

Существуют следующие разновидности циклических алгоритмов:

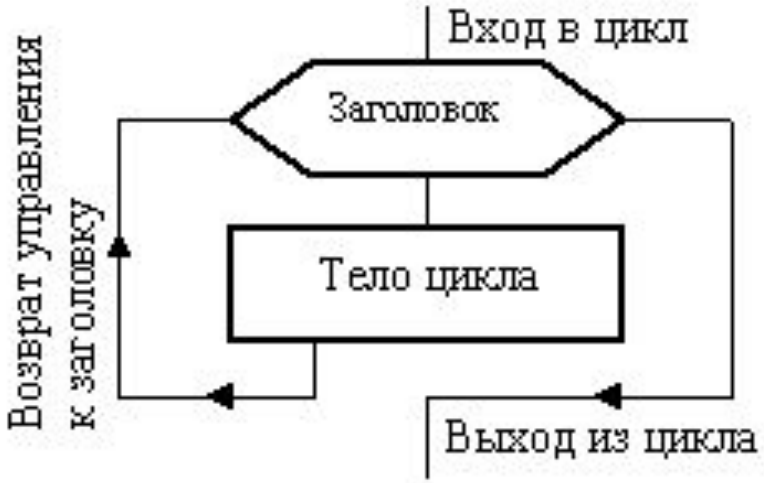
- ***оператор цикла с параметром*** (типа счетчик);
- ***оператор цикла с предусловием***;
- ***оператор цикла с постусловием***.

Если количество повторов известно заранее, используется оператор цикла типа счетчик, т.е. ***детерминированный алгоритм***.

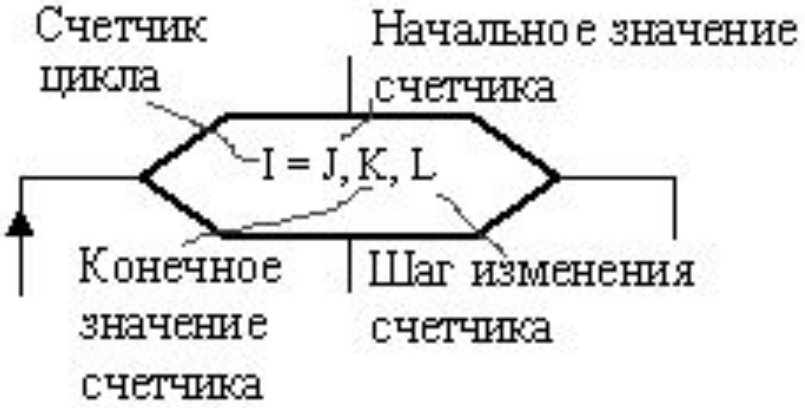
Если количество повторов неизвестно, а задано некоторое условие окончания или продолжения цикла применяются операторы цикла с предусловием или оператор цикла с постусловием. Такие циклы используют для построения ***итерационных алгоритмов***.

Цикл типа счетчик

Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.



Структура цикла

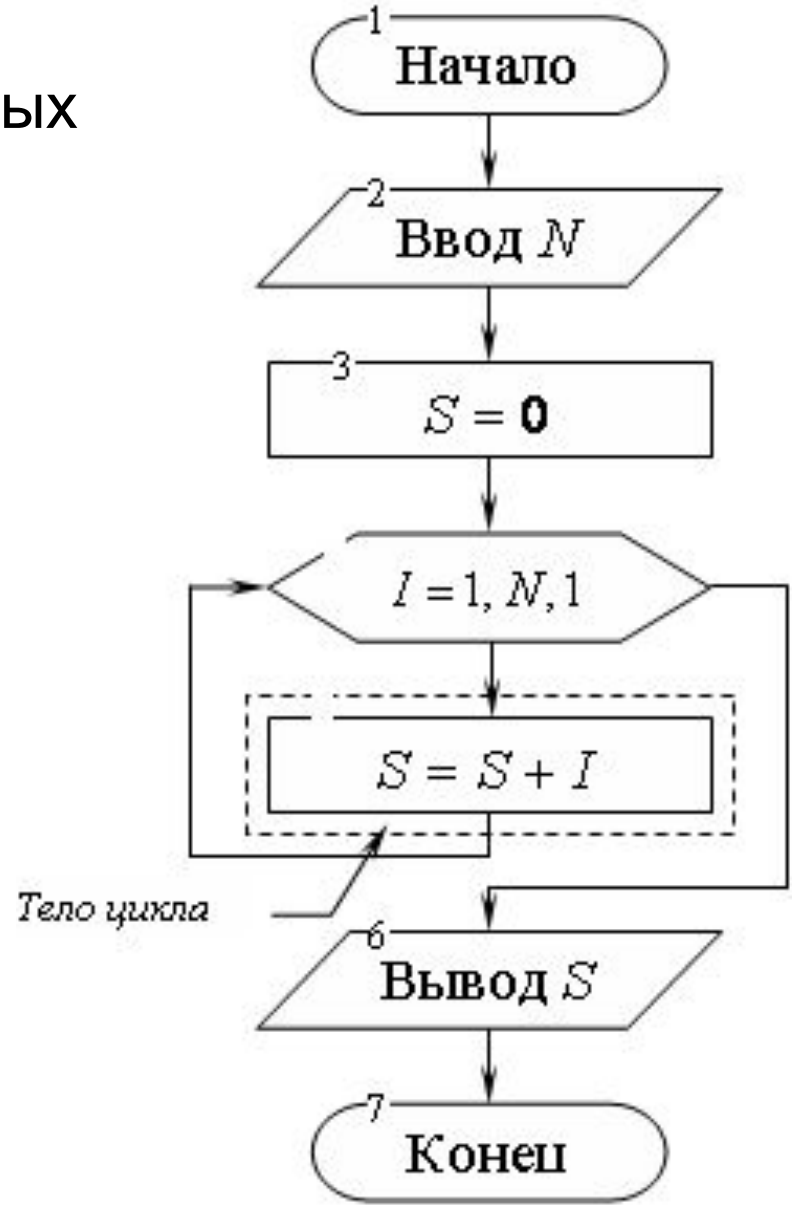


Структура заголовка цикла

Цикл типа счетчик

Блок-схема алгоритма нахождения суммы ***N*** первых натуральных чисел.

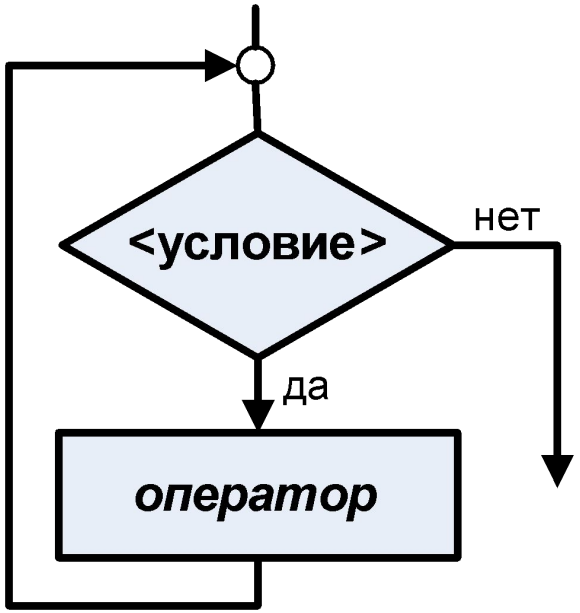
```
Program Cikl;
Var N, I, S: integer;
  Begin
    Writeln('Введите N');
    Read(N);
      S:=0;
       For i:=1 to N do
      S:=S+i;
     Writeln('S= ', S);
  End.
```



Цикл с предусловием

Тело цикла выполняется до тех пор, пока условие истинно.

Если при первой проверке условие оказалось ложным, оператор не выполняется ни разу.



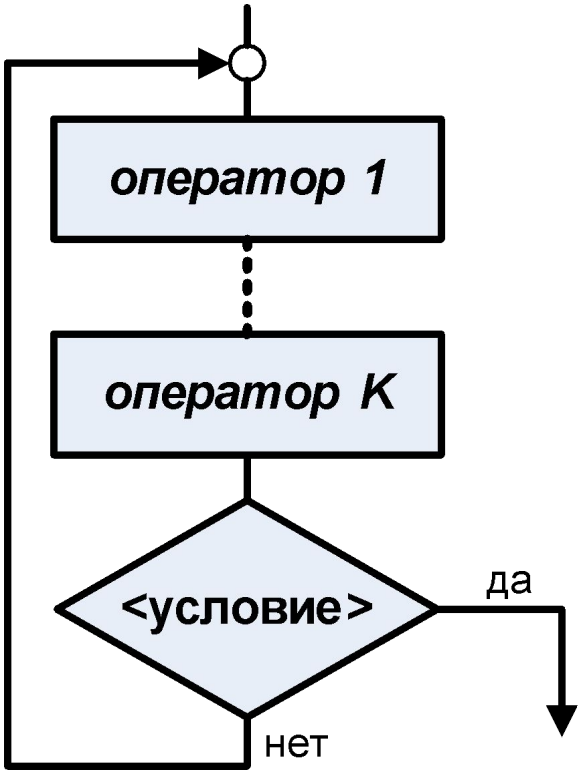
Перед началом цикла должны быть инициализированы переменные, входящие в условие цикла.

Чтобы избежать «зацикливания», среди операторов тела цикла обязательно требуется предусмотреть изменение переменных, входящих в проверяемое условие.

Цикл с постусловием

Оператор используется, когда количество повторений заранее неизвестно, а задано некоторое условие выхода из цикла.

Тело цикла выполняется до тех пор, пока условие ложно.



Если при первой проверке условие оказалось истинным, оператор цикла выполнится один раз.

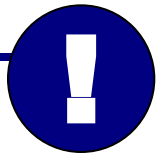
Итерационные циклы

Особенностью итерационного цикла является то, что число повторений операторов тела цикла заранее неизвестно.

Для его организации используются циклы с предусловием или постусловием.

На каждом шаге вычислений происходит ***последовательное приближение к искомому результату и проверка условия достижения последнего***.

Пример 1 итерационного цикла



Составить алгоритм вычисления бесконечной суммы

$$S = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots (-1)^{i-1} \frac{x^i}{i} + \ldots$$

с заданной точностью ε.

Пример 1 итерационного цикла

Особенностью задачи является то, что число слагаемых (а, следовательно, и число повторений тела цикла) заранее неизвестно.

Поэтому выполнение цикла должно завершиться в момент достижения требуемой точности.

Для данной знакочередующейся бесконечной суммы требуемая точность будет достигнута, когда очередное слагаемое, прибавляемое к сумме ряда, станет по абсолютной величине меньше ε.

Пример 1 итерационного цикла

$$S = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots (-1)^{i-1} \frac{x^i}{i} + \ldots$$

Решая эту задачу "в лоб" путем вычисления на каждом *i*–ом шаге частичной суммы

*S:=S + ((–1)^(i–1)) * (x^i) / i* ,

мы получим очень неэффективный алгоритм, требующий выполнения большого числа операций.

Пример 1 итерационного цикла

$$S = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots (-1)^{i-1} \frac{x^i}{i} + \ldots$$

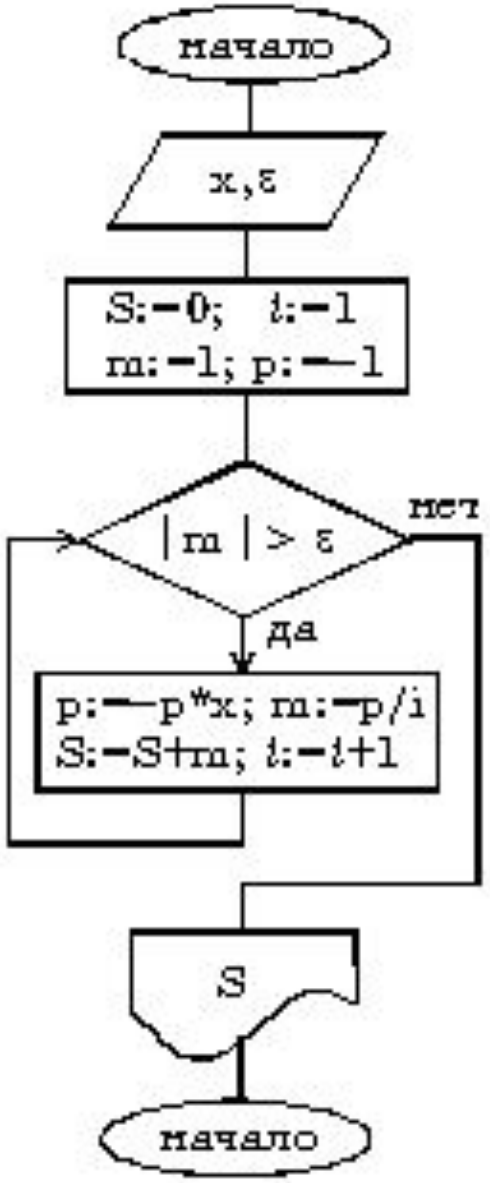
При составлении алгоритма нужно учесть, что знаки слагаемых чередуются, и степень числа ***x*** в числителях слагаемых возрастает.

Гораздо лучше организовать вычисления следующим образом:

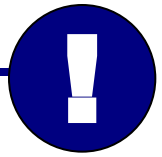
если обозначить числитель какого–либо слагаемого буквой *p*, то у следующего слагаемого числитель будет равен *–p*x* (знак минус обеспечивает чередование знаков слагаемых), а само слагаемое *m* будет равно *p/i* , где *i* – номер слагаемого.

Пример 1 итерационного цикла

$$S = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots (-1)^{i-1} \frac{x^i}{i} + \ldots$$



Пример 2 итерационного цикла



Вычислить значение многочлена (отрезка степенного ряда для e^x):

$$y = \sum_{i=0}^{n} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} = 1 + x + \frac{x^2}{1 \cdot 2} + \ldots + \frac{x^n}{1 \cdot 2 \cdot \ldots \cdot n},$$

Здесь ***x*** и ***n*** - заданные числа.

Пример 2 итерационного цикла

Обозначим общий член ряда через u_i, так что

$$y = \sum_{i=0}^{n} u_i, \text{ где } u_i = \frac{x^i}{i!}.$$

Найдем отношение

$$\frac{u_i}{u_{i-1}} = \frac{x^i \cdot (i-1)!}{i! \cdot x^{i-1}} = \frac{x}{i}, \text{ отсюда } u_i = u_{i-1} \cdot \frac{x}{i}.$$

Таким образом, зная член u_{i-1}, можно вычислить член u_i, выполнив всего две операции.