

Формы представления алгоритмов

- Словесная запись. Недостаток: отсутствие строгой формализации и наглядности вычислительного процесса
- Табличная форма записи алгоритма
- Графовые схемы алгоритмов. (Блок схемы)
- Псевдокод (на псевдоалгоритмическом языке)
- Программная форма записи (тексты на языках программирования)
- Логические схемы алгоритмов – это формальное, математическое описание

Словесная форма представления

Словесная форма записи не широко распространена из-за многословности и отсутствия наглядности.

Пример алгоритма нахождения максимального из двух значений:

Определим форматы переменных X , Y , M , где X и Y – значения для сравнения, M – переменная для хранения максимального значения.

1. Получим два значения чисел X и Y для сравнения;
2. сравним X и Y ;
3. если X меньше Y , значит большее число Y ;
4. поместим в переменную M значение Y ;
5. если X не меньше (больше) Y , значит большее число X ;
6. поместим в переменную M значение X .

Как видно из данного примера словесный способ описания обладает **следующими недостатками**:

- описание строго не формализуемо;
- запись получилась многословной;
- отдельные предписания (действия) допускают неоднозначность толкования.

Приведем словесное описание алгоритма, который вычисляет значение функции $f(x) = \sin x^2 - x$ на интервале $[a, b]$ с шагом $h > 0$. Такая задача называется задачей *табулирования функции* на указанном интервале с определенным шагом.

1. Начало.
2. Задать значения a, b, h .
3. Начиная со значения x , равного a , делать следующее.
4. Вычислить $f(a) = \sin a^2 - a$.
5. Вывести значения a и $f(a)$.
6. Увеличить значение x на шаг и вычислить $f(a+h) = \sin(a+h)^2 - (a+h)$.
7. Вывести значения $a+h$ и $f(a+h)$.
8. Увеличить значение x на шаг и вычислить $f(a+2h) = \sin(a+2h)^2 - (a+2h)$.
9. Вывести значения $a+2h$ и $f(a+2h)$.
10. И так далее продолжать вычислять f , увеличивая x , до тех пор, пока x все еще меньше или равен b , т.е. последнее вычисление такое $f(b) = \sin b^2 - b$.
11. Вывести значения b и $f(b)$.
12. Конец.

Табличная форма записи алгоритма

Табличный способ — это представление алгоритма в виде входных данных, расчётных форм и таблиц.

Способ широко применяется в экономических расчетах. Исходные данные, как и результаты, заносятся в заголовки столбцов используемой таблицы. Простейший пример такого способа представления — таблица умножения.

ТАБЛИЦА УМНОЖЕНИЯ

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Свет в соседней комнате горит	Да	Нет	Нет
Свет у соседей горит	-	Да	Нет
Поменять лампочку	X		
Проверить пробки		X	
Позвонить электрику		X	X
Позвонить диспетчеру			X

Такой способ может быть с успехом применен для проверки правильности функционирования разработанного **алгоритма** на конкретных тестовых наборах входных данных, которые вместе с результатами выполнения **алгоритма** фиксируются в "**таблицах** трассировки".

Графовые схемы алгоритмов (блок-схемы алгоритмов)

Этот метод ещё называют способом блок-схем. В данной ситуации каждый этап прохождения алгоритма представляется в виде геометрических фигур — так называемых «блоков», причём конкретная форма фигур зависит от выполняемой операции. Существует стандарт, регламентирующий размеры используемых графических блоков, а также их отображение, функции, формы и взаимное расположение. Направление работы алгоритма показывают линии соединения блоков.

Другое название способа — **визуальное представление**. При проектировании алгоритмов, представленных графически, придерживаются ряда правил:


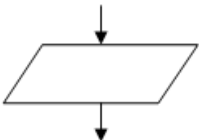
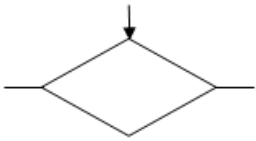
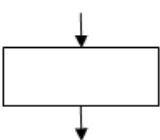
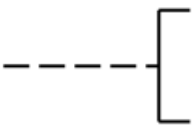
- в начале алгоритма располагаются блоки ввода значений (входные данные);
- после ввода значений располагаются блоки обработки и блоки условия;
- алгоритм завершается блоками вывода значений, полученных в результате работы алгоритма (выходные данные);
- должен быть лишь один блок начала и один — окончания;
- межблочная связь указывается линиями (направленными либо ненаправленными);
- вычислительные формулы, данные и логические выражения размещаются внутри соответствующих блоков;
- возможно наличие комментариев в виде выносок.

Блок-схемы алгоритма

Графический способ представления имеет практическое значение и используется не только в случае программирования. Его применяют при составлении информационных и структурных схем, инфографики и в иных ситуациях, когда нужно обеспечить чёткую визуализацию данных и графически отобразить последовательность расположения объектов алгоритма.

Создание блок-схемы алгоритма — важный и нужный этап решения поставленной задачи. Но при некоторых обстоятельствах этот этап можно считать промежуточным, так как в таком виде описанный алгоритм невозможно выполнить средствами ЭВМ. Зато графический способ представления значительно облегчает процесс дальнейшего создания компьютерной программы.

Основные структурные блоки в схемах алгоритмов

Символ	Название	Назначение
	Терминатор или пуск-останов (начало-конец)	Обозначает начало или конец алгоритма
	Ввод-вывод	Преобразует данные в форму, пригодную для ввода или вывода информации
	Решение или проверка условия	Выбор направления алгоритма в зависимости от некоторых условий
	Процесс (обработка)	Выполнение определенной операции или группы операций
	Комментарий	Сопровождает блок для более детального его описания, если оно не поместилось внутри символа
	Символ-соединитель	Используется для обрыва линии и продолжения ее в другом месте.

Правила оформления блок-схем алгоритмов

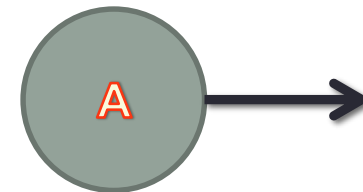
- *Блоки ввода-вывода информации и обработки (выполнения действий) имеют один вход и один выход, блок начала имеет один только выход, а блок конца работы – только вход.*
- *Блок решения (условия) имеет один вход и два выхода, один из которых помечен словом “да” (или 1, или «+» – соответствует переходу при выполнении условия), а второй - словом “нет” (или 0, или «–»).*
- *Последовательность прохождения блоков (передачи управления) определяется стрелками и начинается с блока начала работы. Если направление передачи управления совпадает с естественным (слева-направо, сверху-вниз), то стрелка на дуге, соединяющей смежные блоки, может не ставиться.*

Присваивание: $A \leftarrow B$

«переменная» := «выражение»

Алгоритмы содержат действия по *присваиванию* некоторой переменной значения другой переменной или значения некоторого выражения. Обычно для операции присваивания используется специальное обозначение «:=», или « \leftarrow », или просто «=». Операция присваивания выполняется справа налево, т.е. сначала вычисляется значение выражения **B** в правой части операции присваивания, а затем полученное значение присваивается переменной **A**, стоящей в левой части операции присваивания. Слева от знака присваивания **всегда должна стоять переменная**, которой присваивается значение выражения, стоящего справа. В частном случае слева и справа от знака присваивания может стоять одна и та же переменная. Например, если оператор присваивания имеет вид $x := x + h$ и до его выполнения x , h имеют значения соответственно 6 и 0.5, то после его выполнения h остаётся неизменным, а x будет иметь значение 6.5. Операции присваивания в схемах алгоритмов записываются как правило внутри блоков «**процесс**».

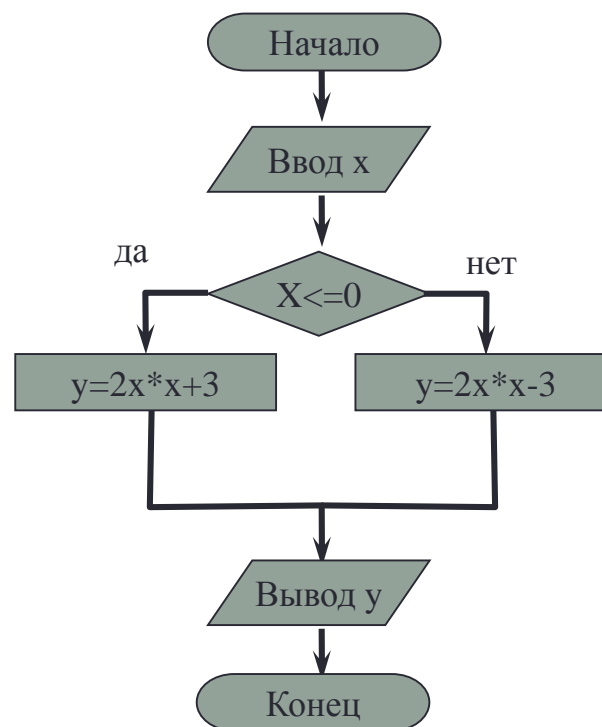
Символ-соединитель



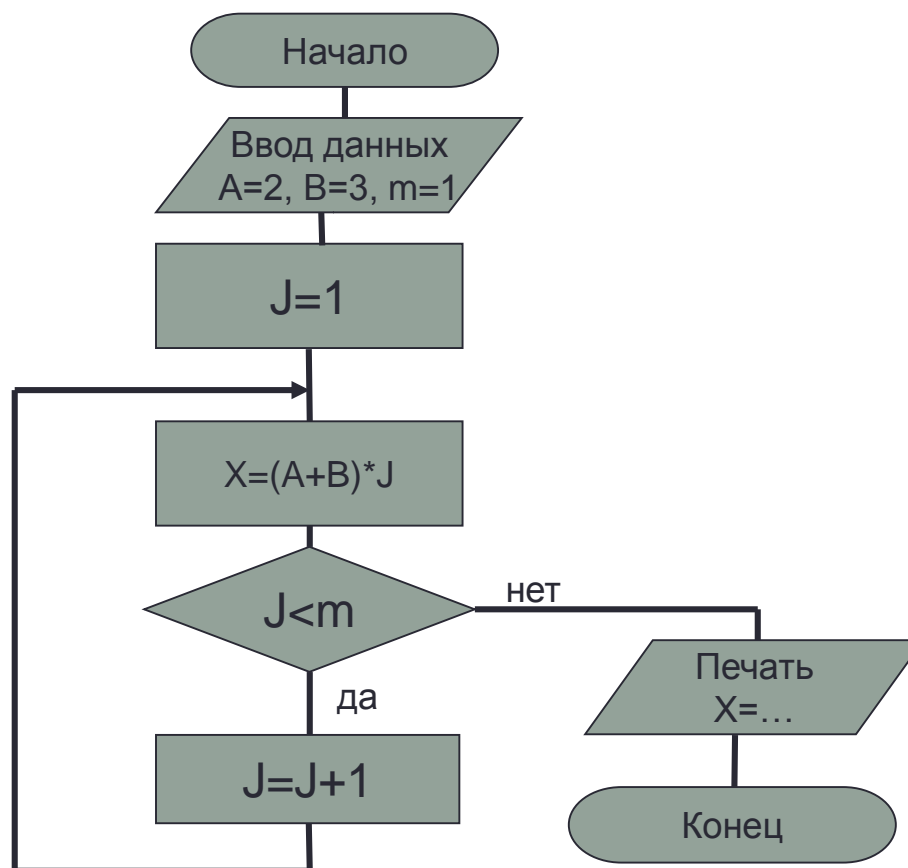
- Если в пределах страницы на схеме неудобно проводить линию передачи управления, то её можно прервать на выходе передающего блока, поставить в точке разрыва кружок с уникальной для этой страницы буквой (именем) внутри, а перед входом принимающего блока поставить кружок с той же буквой и тем самым соединить их. Таким образом, *символ-соединитель* имеет один вход в месте разрыва и один выход в месте соединения. Он отображает выход в часть схемы и вход из другой части схемы. Для каждой «оборванной» линии внутри символа-соединителя должно содержаться одно и то же уникальное обозначение.

Примеры блок-схем

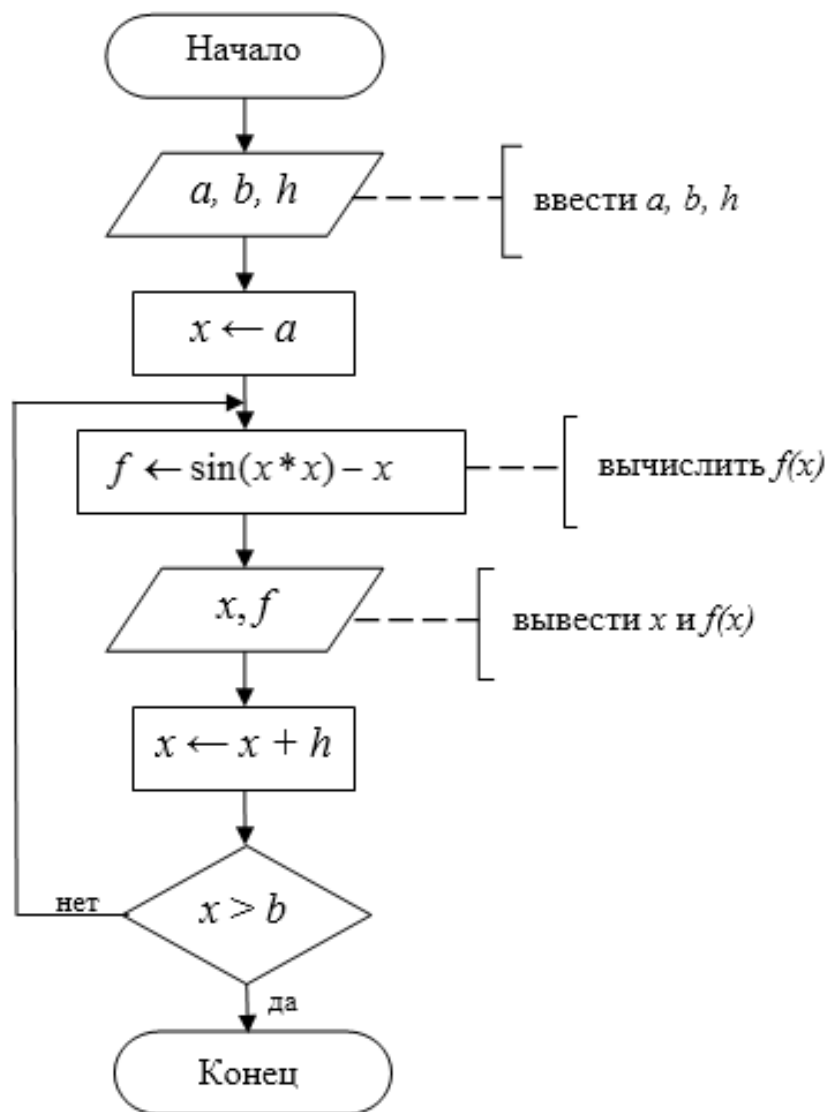
$$y = \begin{cases} 2x^2 + 3, & \text{где } x \leq 0 \\ 2x^2 - 3, & \text{где } x > 0 \end{cases}$$



Примеры блок-схем



- Построим блок-схему алгоритма для решения задачи табулирования функции.



Псевдокод

Псевдокод — компактный язык описания алгоритмов, использующий ключевые слова языков программирования, но опускающий несущественные подробности и специфический синтаксис. Синоним — «псевдо алгоритмический» язык.

Псевдокод обычно опускает детали, несущественные для понимания алгоритма человеком. Такими несущественными деталями могут быть описания переменных, системно-зависимый код и подпрограммы.

Главная цель использования псевдокода — обеспечить понимание алгоритма человеком, сделать описание более воспринимаемым, чем исходный код на языке программирования. Псевдокод широко используется в учебниках и научно-технических публикациях, а также на начальных стадиях разработки компьютерных программ. Блок-схемы и дракон-схемы можно рассматривать как графическую альтернативу псевдокоду.

Примеры псевдокода

Пример. Алгоритм сложения двух чисел:

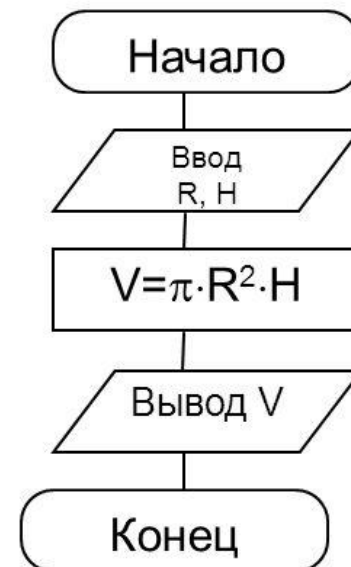
1. Ввод a, b .
2. $S = a + b$.
3. Вывод S .
4. Конец.

Зная радиус основания и высоту цилиндра, вычислить его объем.

Псевдокод:

1. Ввод R и H .
2. $V = \pi \cdot R^2 \cdot H$.
3. Вывод V .
4. Конец.

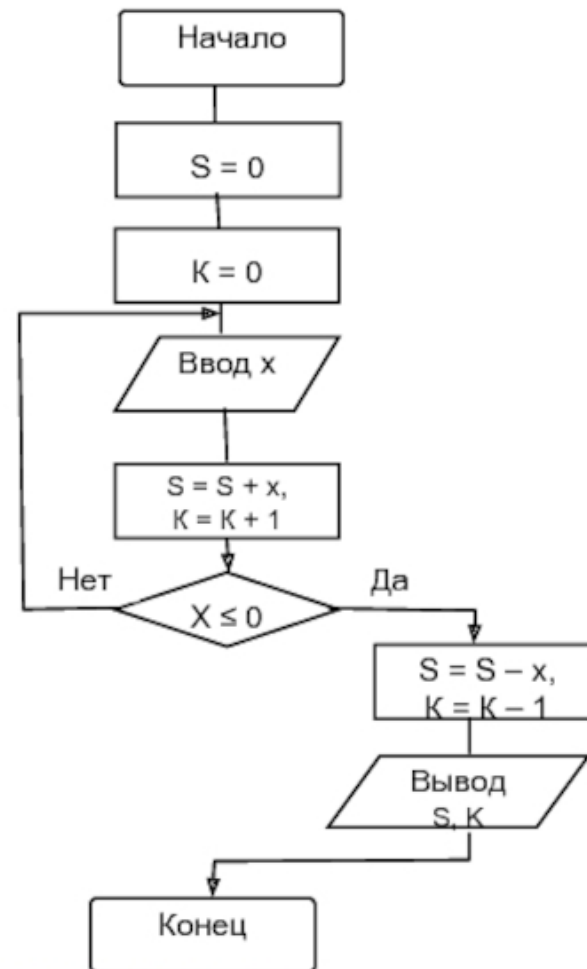
Блок-схема:



Примеры псевдокода

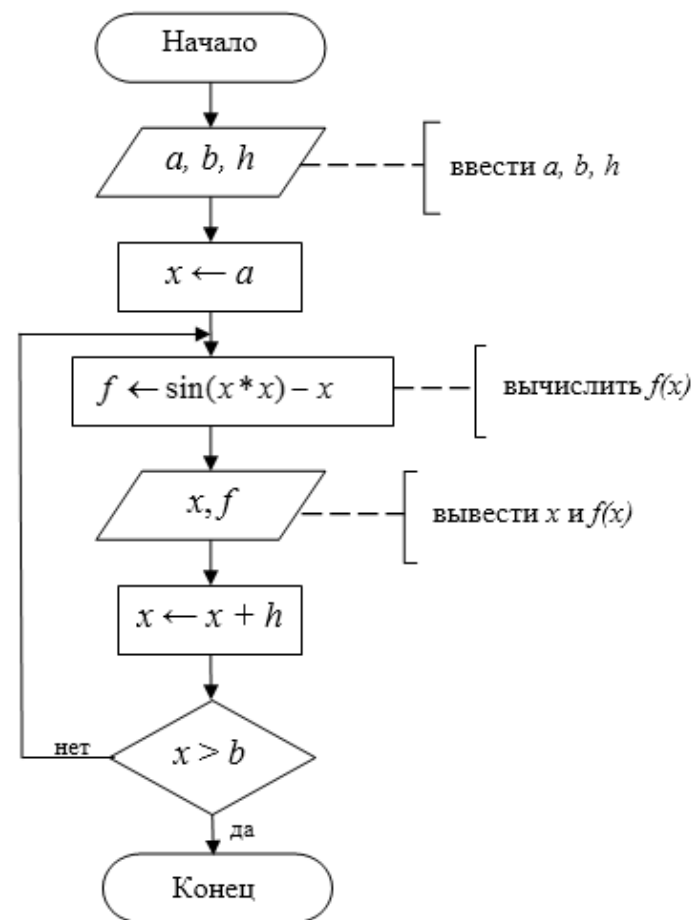
Псевдокод:

1. $S = 0, K = 0$.
2. Ввод x .
3. $S = S + x$.
4. $K = K + 1$.
5. Если $x \leq 0$, ТО переходим к шагу 6. ИНАЧЕ переходим к п. 2.
6. $S = S - x, K = K - 1$.
7. Вывод S, K
8. Конеч.



- Приведем запись алгоритма на псевдокоде для решения задачи табулирования функции. Алгоритм должен соответствовать разработанной ранее блок-схеме.

1. Начать работу.
2. Ввести значения a, b, h .
3. $x \equiv a$ (присвоить x значение a).
4. $f(x) := \sin(x * x) - x$ (вычислить $f(x)$).
5. Вывести x и $f(x)$.
6. $x \equiv x + h$ (увеличить x на значение h).
7. Если $x \geq b$, то перейти к п.8, иначе перейти к п.4.
8. Конец работы.



Программы

Н. Вирт
АЛГОРИТМЫ+
СТРУКТУРЫ ДАННЫХ=
ПРОГРАММЫ

- **Программа** – формальная запись алгоритма на одном из языков программирования.
- Программа представляет собой алгоритм, который записан как последовательность команд. Речь идёт о командах, понятных компьютеру, для чего используются различные языки программирования, представляющие собой системы кодирования предписаний с правилами их применения.
- Языки программирования характеризуются строго определённым синтаксисом, то есть свободное толкование конструкций не допускается.
- В случае программного способа представления алгоритмическая последовательность записывается в виде компьютерной программы с высокой степенью формализации. В результате появляется возможность решать прикладные задачи.
- **Компьютерная программа** — последовательность инструкций, предназначенная для исполнения устройством управления вычислительной машины.

Программы (примеры)

Задача. По введенному числу X вычислить значение функции

$$y = \frac{x^2 - 3}{\sqrt{x} + 1}$$



Язык Pascal

```
Program example;
Var x,y: real;
Begin
  writeln(' Введите на клавиатуре число X ');
  readln(x);
  y := (x*x-3)/(sqrt(x)+1);
  writeln(' y = ' , y:6:3);
End.
```



Язык C#

```
using System;

double x, y;
Console.Write("Введите на клавиатуре число X: ");

x = Convert.ToDouble(Console.ReadLine());

y = (x * x - 3) / (Math.Sqrt(x) + 1);

Console.Write("y = "+ y);
```



Язык Python

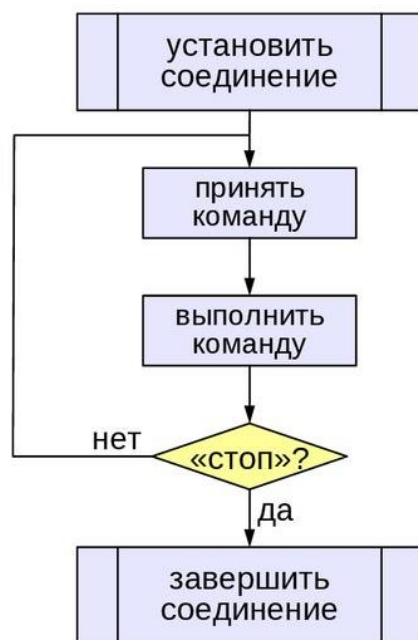
```
import math
x=float(input('Введите на клавиатуре число X: '))
y = (x*x-3)/(math.sqrt(x)+1)
print('y = ',y)
```

Результат выполнения программы

```
Введите на клавиатуре число X: 6
y = 9.566632302368976
```

Запись алгоритма в виде программы

блок-схема



программа

```
setConnection;  
repeat  
  cmd:= getCommand;  
  executeCommand(cmd);  
  
until cmd = "stop";  
closeConnection;
```



Логические схемы алгоритмов

Формальный язык для описания алгоритмов (ЛСА) был первоначально ориентирован на задачи оптимизации программ для ЭВМ, а в последствии оказался удобным и для оптимизации микропрограмм. Задачи этих классов широко распространены как в области проектирования средств вычислительной техники, так и при построении систем автоматического управления, связи.

Реализация алгоритма - последовательное выполнение команд, каждая из которых является последовательностью элементарных действий-микрокоманд, выполняемых за один машинный такт.

Функцию задания алгоритма А.А.Ляпунов предложил записывать в виде конечной строки, состоящей из символов элементарных действий A_i , где i изменяется от 1 до n (n – целое положительное число), логических функций α_r ($r = 1, m$) и специальных символов начала и конца стрелок с индексом, где индекс также целое положительное число.

Пример записи алгоритма в виде логической схемы

$$\Pi_q \uparrow^t \Psi_r \dots \Psi_s \downarrow^q \Pi_t \dots \text{ОСТ}$$

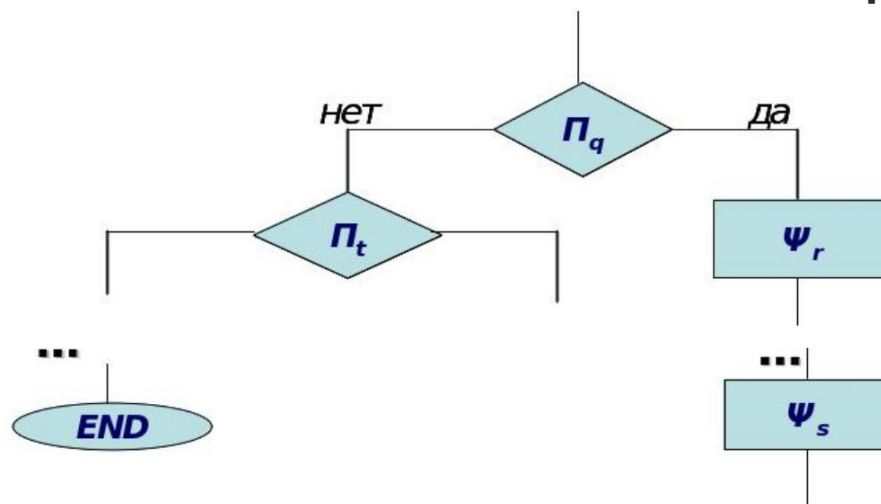
Здесь q -й безусловный оператор или составной безусловный оператор обозначается через Ψ_q , а r -й оператор условного перехода или составной оператор условного перехода, заканчивающийся оператором типа «если ..., то ..., иначе ...», – через Π_r . Переходы в алгоритме отображаются стрелками. После выполнения очередного оператора и при отсутствии перехода выполняется следующий справа оператор. Переход задается точкой выхода (стрелка вверх) и точкой входа (стрелка вниз). Безусловный переход от k -му оператору имеет обозначение \uparrow^k , расположенное непосредственно за Ψ_q , и обозначение \downarrow^q , расположенное непосредственно перед обозначением r -го оператора. Оператор Π_q , оканчивающийся оператором условного перехода, например, «если ДА, то переход к оператору Ψ_r , иначе переход к оператору Π_t », в случае, когда оператору в записи алгоритма предшествует оператор Ψ_s , отображается записью $\uparrow^t \dots \downarrow^q$. Окончание работы алгоритма отмечается сокращением ОСТ, обозначающим оператор останова.

Действие каждого оператора затем описывается с требуемой степенью детализации.

Примеры записи алгоритмов в виде логической схемы

Пример 1. $\Pi_q \uparrow^t \Psi_r \dots \Psi_s \downarrow^q \Pi_t \dots \text{ОСТ}$

Эквивалентная блок-схема алгоритма



Пример 2.

$\Psi_1 \downarrow^{17} \Pi_2 \uparrow^4 \Psi_3 \text{ ОСТ } \downarrow^2 \Psi_4 \downarrow^8 \Psi_5 \downarrow^{13} \Pi_6 \uparrow^{12} \Pi_7 \uparrow^9 \Psi_8 \uparrow^5 \downarrow^7 \Pi_9 \uparrow^{11} \Psi_{10} \uparrow^{26} \downarrow^9 \Psi_{11} \text{ ОСТ } \downarrow^6 \Psi_{12} \downarrow^{14} \downarrow^{22} \Pi_{13} \uparrow^6 \Pi_{14}$
 $\uparrow^{13} \Psi_{15} \downarrow^{20} \Pi_{16} \uparrow^{18} \Psi_{17} \uparrow^2 \downarrow^{16} \Pi_{18} \uparrow^{21} \downarrow^{24} \Pi_{19} \uparrow^{25} \downarrow^{29} \Psi_{20} \uparrow^{16} \downarrow^{18} \Psi_{21} \downarrow^{23} \downarrow^{26} \Pi_{22} \uparrow^{13} \Pi_{23} \uparrow^{22} \Psi_{24} \uparrow^{19} \downarrow^{19} \Psi_{25} \downarrow^{10}$
 $\downarrow^{27} \Pi_{26} \uparrow^{22} \Pi_{27} \uparrow^{26} \Psi_{28} \Psi_{29} \uparrow^{20}.$

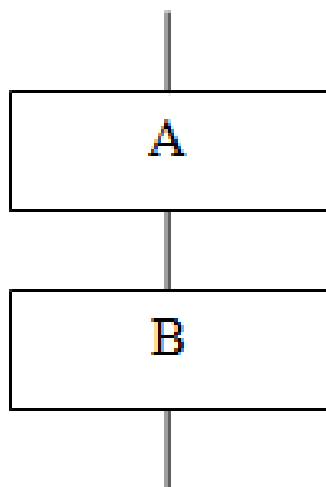
БАЗОВЫЕ СТРУКТУРЫ АЛГОРИТМА

Структурные алгоритмы

- Для решения прикладной задачи всегда **можно составить несколько разных алгоритмов**.
- Из этих возможных алгоритмов нужно выбрать самый хороший для дальнейшей программной реализации.
- Чтобы оценить насколько «хорош» алгоритм, анализируются следующие характеристики:
 - простота и легкость понимания алгоритма,
 - скорость выполнения и требуемый объём памяти.
- Наиболее важными в настоящее время являются простота и легкость понимания. Причина этого – большое количество создаваемых алгоритмов и программ, необходимость обмена алгоритмами и программами между людьми, предприятиями, организациями и государствами. Для того чтобы алгоритм был простым и легко понимаемым, он должен быть структурным и его рекомендуется строить, используя 3 основных структуры

1. Последовательность (последовательное соединение)

Выполняется несколько последовательных команд (операторов). На рисунке они обозначаются буквами А и В.



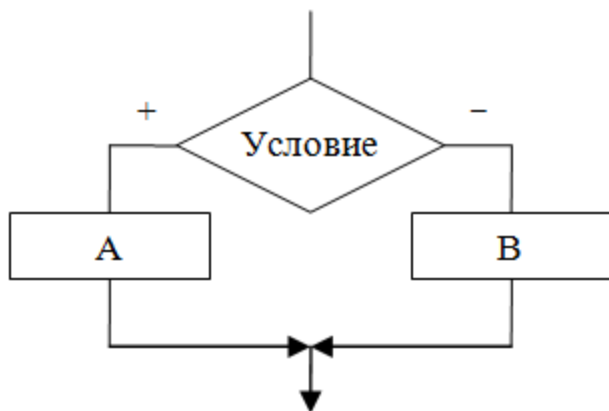
На алгоритмическом языке	На языке программирования	
	Си	Паскаль
A; B;	{ A;B; }	begin A; B end

На языке Python
A B

2. Условие (проверка условия).

2.1. Условие с двумя вариантами действий.

- Если истинно условие, то выполняется блок (оператор) А, иначе выполняется блок В (оператор В)



На алгоритмическом языке	На языке программирования	
	Си	Паскаль
если <i>Условие</i> то А, иначе В	if (<i>Условие</i>) А else В;	if <i>Условие</i> then А else В;

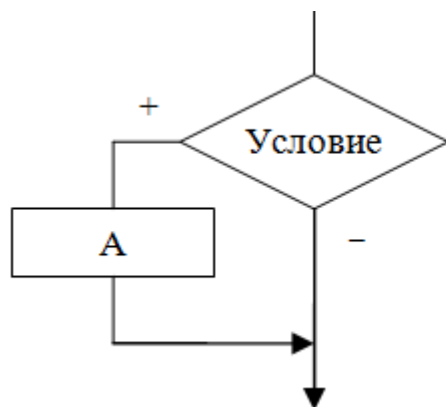
На языке Python

```

if Условие:
    А
else:
    В
  
```

2.2. Условие с одним вариантом действий.

- Если истинно условие, то выполняется блок (оператор) А, иначе ничего не выполняется и управление передается следующему оператору



На алгоритмическом языке	На языке программирования	
	Си	Паскаль
если <i>Условие</i> то А	if (<i>Условие</i>) А;	if <i>Условие</i> then А;

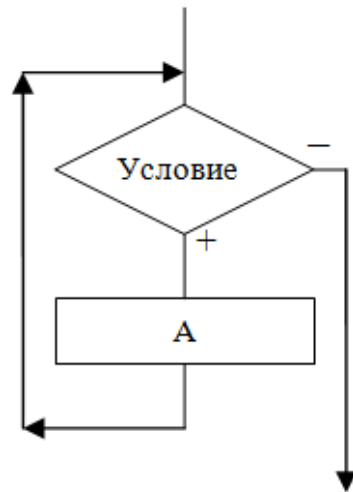
На языке Python

```
if Условие:
    А
```

3. Цикл (циклическое повторение).

3.1. Цикл с предусловием.

- В начале цикла проверяется условие и после этого циклически выполняется некоторое действие А (блок операторов) до тех пор, пока условие все еще верно (истинно). Как только условие становится ложным, цикл завершается



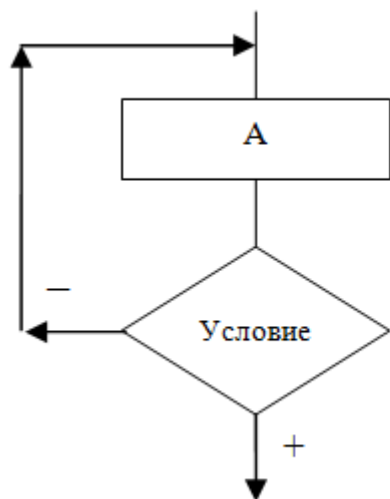
На алгоритмическом языке	На языке программирования	
	Си	Паскаль
Начало цикла пока <i>Условие</i> выполнять <i>А</i> ; Конец цикла	<code>while (<i>Условие</i>) { <i>А</i>; }</code>	<code>while <i>Условие</i> do begin <i>А</i> end;</code>

На языке Python

```
while (Условие) :  
  А
```

3.2. Цикл с постусловием.

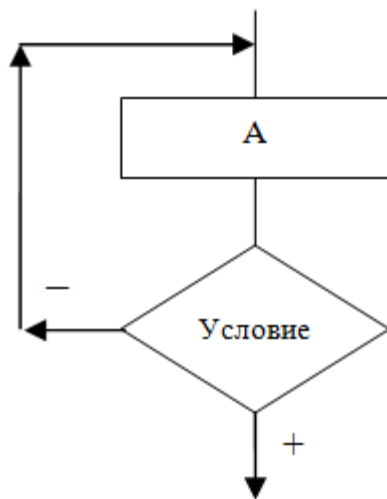
- Некоторое действие A (блок операторов) повторяется до тех пор, пока условие в конце цикла не станет верным (истинным). Как только условие в конце цикла становится истинным, цикл завершается



На алгоритмическом языке	На языке программирования	
	Си	Паскаль
Начало цикла повторять A до <i>Условие</i>	do { A; } while (<i>Условие</i>);	repeat A until <i>Условие</i> ;

К сожалению, в Python цикл с постусловием отсутствует!

Использование операторов **while** и **break** для реализации цикла с пост-условием



На языке Python

```
while True :  
    A  
    if Условие:  
        break
```

Примечание

- Во всех трёх случаях, во всех стандартных структурах блоки **A** и **B** могут в свою очередь состоять из таких же стандартных структур. Т.е. одна структура может быть вложена в другую произвольное количество раз.

Структурный алгоритм, программа, структурное программирование, теорема о структурных алгоритмах

- **Структурный алгоритм** – это алгоритм, в котором используются только стандартные структуры – последовательность, условие, цикл, а также ввод-вывод.
- **Структурная программа** – это программа, которая точно соответствует структурному алгоритму.
- **Структурное программирование** – процесс разработки структурных алгоритмов и программ.
- Справедлива следующая **теорема о структурных алгоритмах**: для каждого алгоритма существует эквивалентный ему структурный алгоритм.

Пример.

- Напишем два варианта структурных алгоритмов решения задачи табулирования функции, на основе циклов с постусловием (слева) и предусловием (справа)

<p>0. Начало</p> <p>1. Ввод a, b, h</p> <p>2. $x := a$</p> <p>3. Начало цикла</p> <p>Повторять</p> <p>3.1. $f := \sin(x * x) - x$</p> <p>3.2. Вывод x, f</p> <p>3.3. $x := x + h$</p> <p>3.4. до $x > b$</p> <p>Конец цикла</p> <p>4. Конец.</p>	<p>0. Начало</p> <p>1. Ввод a, b, h</p> <p>2. $x := a$</p> <p>3. Начало цикла</p> <p>Пока $x \leq b$ выполнять</p> <p>3.1. $f := \sin(x * x) - x$</p> <p>3.2. Вывод x, f</p> <p>3.3. $x := x + h$</p> <p>Конец цикла</p> <p>4. Конец.</p>
--	--

Безусловный переход

- В некоторых языках программирования отсутствуют полноценные структуры для реализации условий и циклов. Поэтому в дополнение к трём стандартным структурам иногда применяется *безусловный переход*. При выполнении безусловного перехода управление передается на указанный шаг (пункт) в алгоритме

Переход (безусловный переход)		
На алгоритмическом языке	На языке программирования	
	Си	Паскаль
Перейти к пункту N	goto N;	goto N;

К счастью, в Python оператор goto отсутствует!

Ограничение использования безусловного перехода

- Использование безусловного перехода приводит к нарушению принципов структурного программирования. Поэтому его применение в структурном программировании – ограничено. Однако существуют случаи, когда без него не обойтись, например при обработке критических ошибок. В других ситуациях хороший стиль программирования полностью исключает использование этой конструкции. Считается плохим тоном писать структурные программы с использованием безусловного перехода. Следует также отметить, что в языках машинного (низкого) уровня типа Ассемблер без этих конструкций обойтись невозможно.

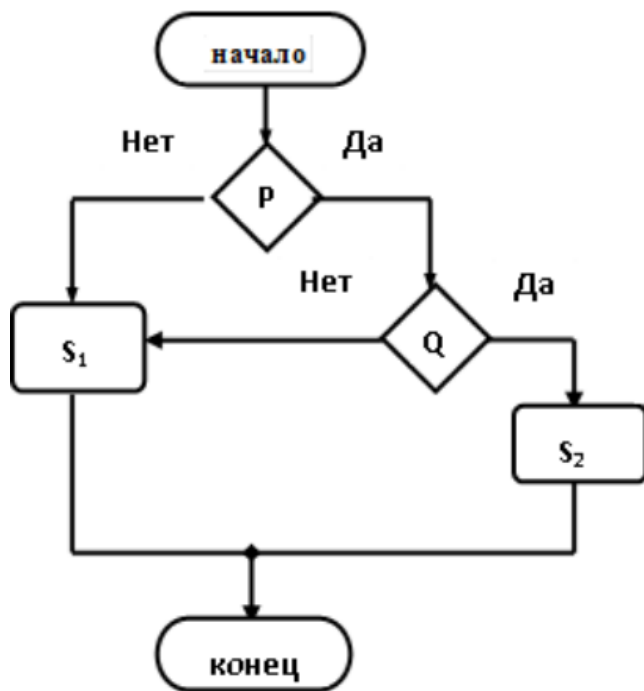
Преобразование неструктурных алгоритмов в структурные

Структурный алгоритм не всегда удастся построить сразу. Поэтому в таких случаях можно поступать следующим образом:

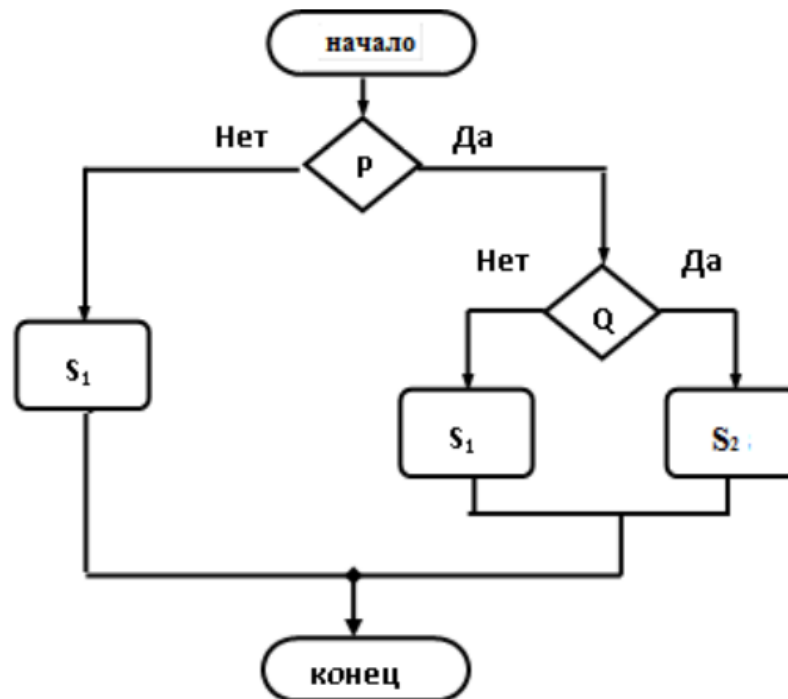
- Сначала построить неструктурный алгоритм, решающий поставленную задачу;
- Затем с помощью специальных приемов, преобразовать его в структурный.

Таких приемов существует много, рассмотрим некоторые из них: дублирование отдельных блоков схемы, объединение нескольких простых условий в одно сложное, использование переменной флажка.

Дублирование отдельных блоков схемы



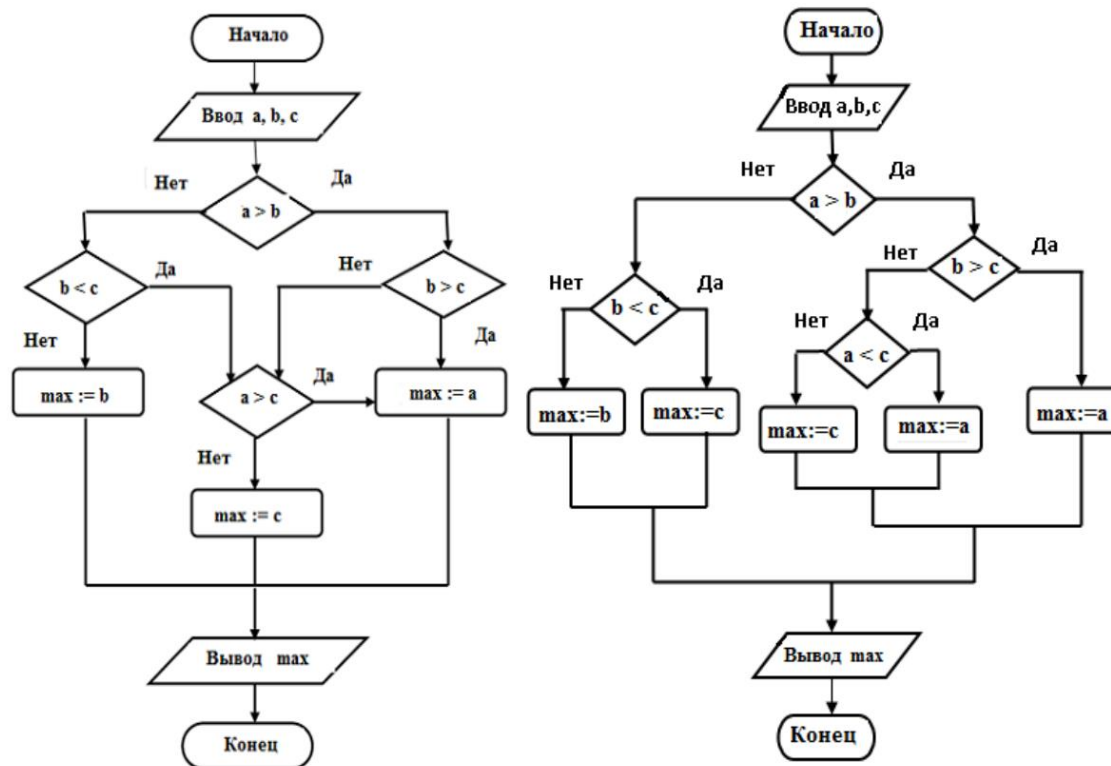
Неструктурная блок-схема



Структурная блок-схемы

Дублирование отдельных блоков схемы

Пример. Составить блок – схему решения задачи: заданы три переменные a , b , c . Определить, какая из них имеет максимальное значение.



Объединение условий

Пример . Имеется набор из n заданных чисел: $x_1, x_2, x_3, \dots, x_n$.

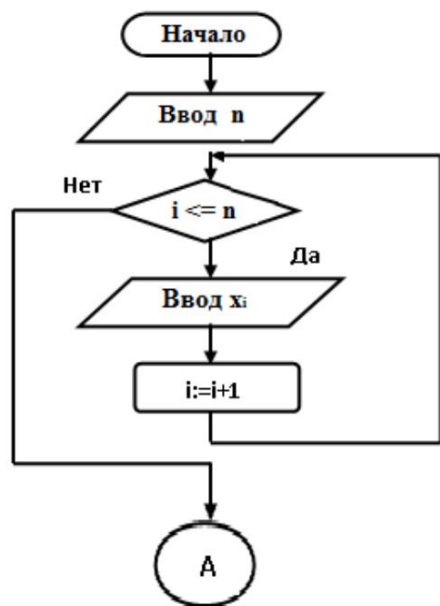
Определить, имеется ли в этом наборе хотя бы одна пара соседних взаимнообратных чисел.

Рассмотрим идею решения задачи. Известно, что если два числа x_i и x_{i+1} являются взаимно обратными, то их произведение $x_i \cdot x_{i+1} = 1$, и наоборот, если числа не взаимно обратные, то их произведение $x_i \cdot x_{i+1} \neq 1$.

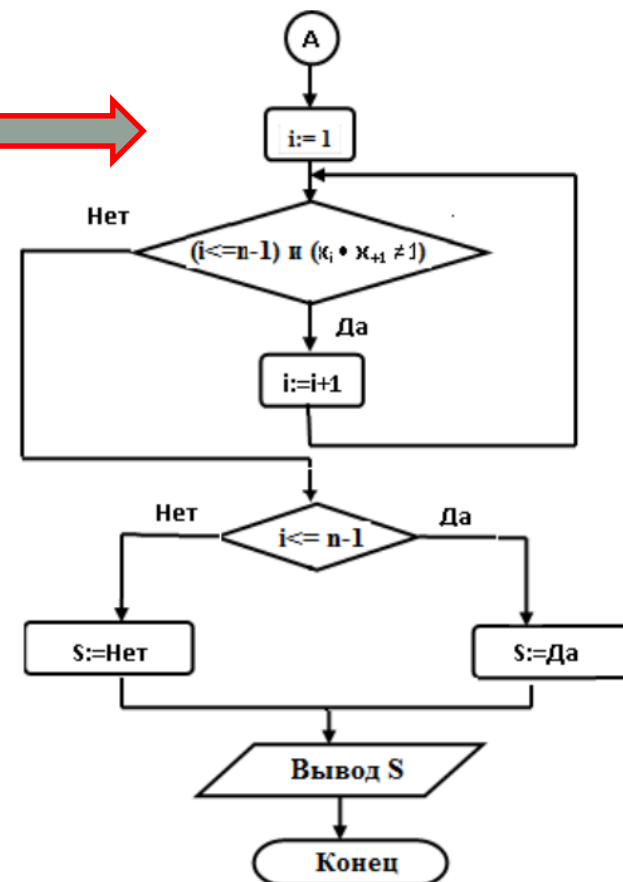
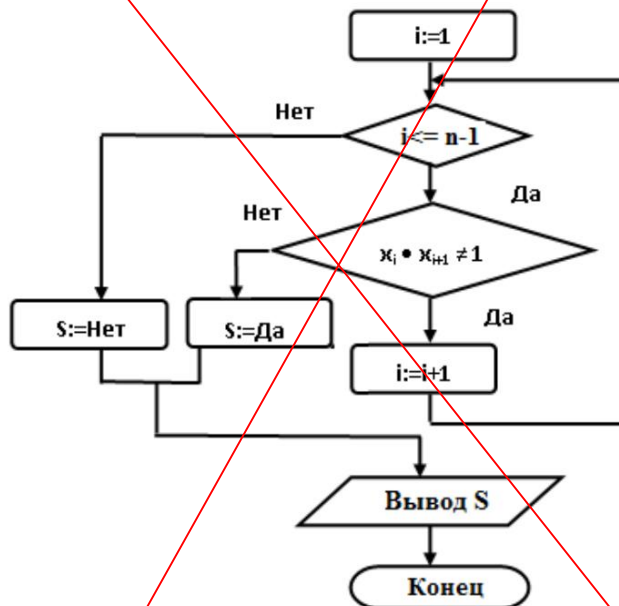
В алгоритме будем использовать второе неравенство, то есть, будем циклически проверять выполнение неравенства $x_i \cdot x_{i+1} \neq 1$ для $i = 1, 2, \dots, n-1$.

Как только это неравенство станет ложным (то есть, когда $x_i \cdot x_{i+1} = 1$), цикл будем заканчивать, в этом случае ответ на вопрос задачи – «Да».

Если же для всех пар чисел неравенство $x_i \cdot x_{i+1} \neq 1$ будет истинным, то ответ будет – «Нет».



Блок-схема решения задачи



Структурная часть блок-схемы задачи

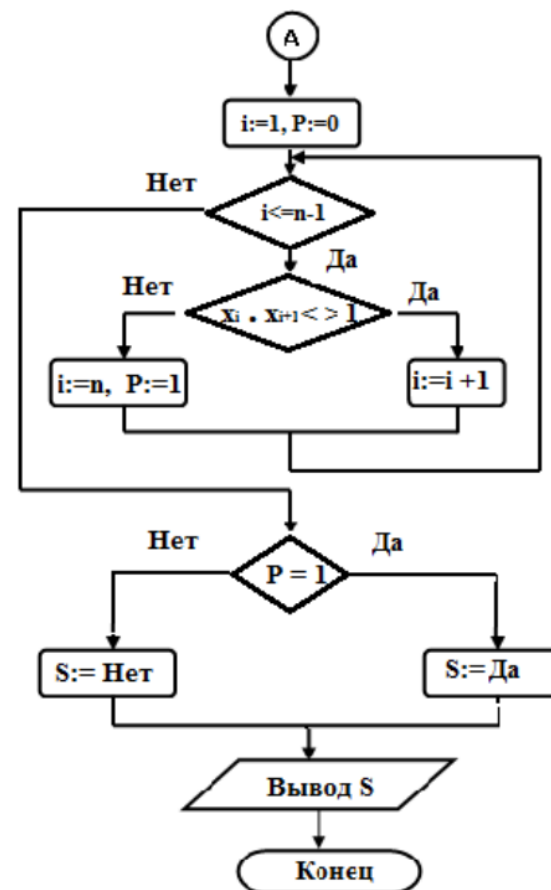
Использование переменной флажка

Иногда блок–схему удастся превратить в структурную с помощью использования дополнительной переменной, которая принимает определенное значение при выполнении какого-либо условия.

Такую переменную называют «флажком».

Используем этот прием для структурирования неструктурной части блок – схемы из предыдущего примера.

Флажок обозначим символом P .



Блок-схема с переменной флажком

Программирование (кодирование) или программная реализация алгоритмов

- Словесную или графическую запись алгоритма обычно нельзя сразу ввести в ЭВМ, поэтому необходимо записать алгоритм на каком-либо языке программирования (ЯП). В результате получается программа на ЯП, которая вводится на ЭВМ и поступает на обработку в системную программу – транслятор (переводчик). *Транслятор* проверяет программу и выдаёт пользователю сообщение об ошибках. Если в программе ошибок нет, то транслятор переводит программу с ЯП на внутренний машинный язык ЭВМ. В результате получается машинная программа, которая управляет работой ЭВМ в процессе решения прикладной задачи. Переход от алгоритма к программе на ЯП называется *кодированием* алгоритма.

Неоднозначность кодирования алгоритмов

- Для каждого алгоритма можно построить несколько вариантов программы, как на одном, так и на разных языках программирования.
- Поэтому при кодировании алгоритма нужно оптимизировать программу по лёгкости понимания, быстродействию и объёму памяти.
- Соответственно и программы (код) могут получаться «хорошие» и «плохие», «понятные» и «непонятные», «длинные» и «короткие», «быстрые и медленные» и т.д.
- Искусство кодирования(программирования) — писать программы лучшие по всем характеристикам

Процесс кодирования

- С технической точки зрения процесс кодирования алгоритма заключается в записи основных алгоритмических конструкций на языке программирования. Очень просто это можно представить в виде таблицы, в которой сводятся основные алгоритмические конструкции и соответствующие им конструкции языка программирования
- Тогда процесс кодирования сводится к простой подстановке вместо структурных блоков алгоритма соответствующих команд языка программирования

Пример таблицы кодирования алгоритмов на языке С# и Паскаль

№	Язык С#	Язык Паскаль
1	Блок начало	
	private static void Main(string[] args){	PROGRAM Name; BEGIN
2	Блок конец	
	}	END.
3	Операция присваивания: переменной x присвоить значение Z: $x \leftarrow Z$	
	x = Z ;	x := Z;
4	Разделение команд (операторов). Символ разделитель	
	;	;
5	Ввод данных	
	double F = Convert.ToDouble(Console.ReadLine()); int A = Convert.ToInt32(Console.ReadLine());	READ (перемен1, перемен2,...); или READLN (перемен1, перемен2,...);
6	Вывод данных	
	Console.Write(""+перемен1+ перемен2+...); или Console.WriteLine(""+перемен1+ перемен2+...);	WRITE (перемен1, перемен2,...); или WRITELN (перемен1, перемен2,...);
7	Условие с двумя вариантами действий	
	if (Условие) { A;} else {B;}	IF (Условие) THEN BEGIN A; END ELSE BEGIN B; END;
8	Условие с одним вариантом действий	
	if (Условие) { A;}	IF (Условие) THEN BEGIN A; END;

Продолжение таблицы кодирования

9	Цикл с предусловием	
	while (Условие) { A; }	WHILE (Условие) DO BEGIN A; END
10	Цикл с постусловием	
	do { A; } while (Условие) ;	REPEAT A; UNTIL (Условие) ;
11	Описание данных	
	Main() { int i, j,...; // <i>целые данные</i> double x,y,z,...; // <i>вещественные</i> char c,p,...; // <i>символьные</i> string str; // <i>строковые</i> ... }	PROGRAM FIRST; VAR i, j,... : INTEGER ; {целые данные} x,y,z,... : REAL ; {вещественные} c,p,... : CHAR; {символьные} str : STRING; {строковые} BEGIN ... END
12	Подключение библиотек и модулей	
	//using System; internal class Program {...	{В случае необходимости} UNIT имямодуля-библиотеки; PROGRAM Name2; ... END
13	Комментарий	
	// текст в одной строке или /* многострочный текст */	{текст комментария в одну или несколько строк} или (* текст в одну или несколько строк *) или // текст в одной строке

Пример.

- Используя таблицу, закодируем на языках Си# и Паскаль структурные алгоритмы решения задачи табулирования функции на основе цикла с постусловием (на языке Си#) и цикла с предусловием (на языке Паскаль).

Пример кодирования алгоритма

```
using System;
internal class Program
{
    private static void Main(string[]
args)
    { double a,b,h,x,f ;
      a =
Convert.ToDouble(Console.ReadLine());
      b =
Convert.ToDouble(Console.ReadLine());
      h =
Convert.ToDouble(Console.ReadLine());
      x = a;
      do {
          f = sin(x*x) - x;
          Console.WriteLine(" " + x + f );
          x = x+h;
      } while ( x>b );
    }
}
```

```
PROGRAM TABUL1;
VAR
    a,b,h,x,f: REAL;
BEGIN
    READLN (a, b, h);
    x := A;
    WHILE ( x<= B ) DO
    BEGIN
        f:=sin(x*x) - x;
        WRITELN(x, f);
        x:=x+h;
    END
END.
```

Разбор задачи по разработке алгоритма

Рассмотрим пример составления алгоритма для вычисления алгебраического выражения с параметрами. В качестве параметра будет выступать переменная, увеличивающаяся на единицу и принимающая новое значение на каждом следующем итерационном шаге.

Задание:

$$y = \frac{\sqrt{x^2 - x}}{\log_2 x + \log_x 2}$$

При

$$x_i = 1, 2, \dots, 100$$

Здесь переменная x_i означает, что существует такое целое количество чисел X , которое укладывается на интервале от 1 до 100 с шагом 1, где i означает индекс каждого из чисел.

Результатом выражения будет y_i для каждого числа x_i .

Разбор задачи по разработке алгоритма

Для того, чтобы решить данное задание, нам необходимо определиться с тем, какие стандартные блоки схем алгоритмов мы будем использовать при его выполнении.

1. Для начала нам необходимо обозначить место, где наш алгоритм будет начинаться. Это можно сделать при помощи специализированного блока, содержащего заголовок «Начало».

Начало

2. После этого, нам необходимо определиться, что должно вводиться в программу в качестве аргумента. Для нашей задачи наилучшим образом будет принять допущение, что все элементы всегда начинаются с целого числа «1», каждое следующее число увеличивается на единицу, а последнее число является тем целым положительным числом, которое пользователь ввел с клавиатуры. Таким образом в качестве аргумента предлагается использовать « i », которое мы будем вводить. Таким образом, следующий блок, который мы получим, должен быть блок ввода, содержащий в качестве аргумента целое положительное число « i ».

Ввод i

3. После необходимо ввести переменную, которая будет хранить в себе номер элемента и

Проинициализировать ее значением $n = 1$.

$n = 1$

Разбор задачи по разработке алгоритма

Для начала выведем формулу по которой будет считаться x_i . Чтобы при вводе целого числа с клавиатуры, на каждом шаге расчета получить актуальное число x_i , необходимо воспользоваться формулой: $x_i = 1 * n$. Тогда, используя формулу, проверим чему будет равняться 10й элемент последовательности чисел, подставив число 10 в полученное уравнение на место переменной n , означающее номер элемента соответствующего шагу расчета: $x_{10} = 1 * 10 = 10$.

4. После блока ввода данных, нам необходимо добавить блок вычисления x_i :

$$x_i = 1 * n$$

5. Следующим шагом мы можем добавить блок в котором будет вычисляться само уравнение:

$$y = \frac{\sqrt{x^2 - x}}{\log_2 x + \log_x 2}$$

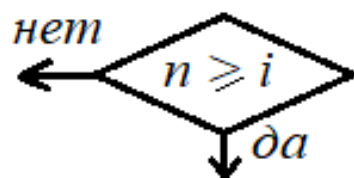
6. Поскольку нам необходимо вывести результаты всех вычислений для каждого из x_i , то в конце каждой операции нам необходимо добавить вывод результата значения y_i и x_i .

Вывод y_i и x_i

Разбор задачи по разработке алгоритма

7. Далее необходимо проверить, если число i , введенное с клавиатуры, совпадает с количеством шагов, то необходимо окончить расчет и завершить программу, а если не совпадает и меньше, то увеличить число n на ещё один шаг.

Для решения этой задачи нам потребуется блок проверки условия, который будет содержать проверку $n \geq i$:



8. После проверки условия, если оно не верное, происходит прибавление единицы к переменной n :

$n = n + 1$

9. Если условие оказывается истинным, выполнение программы завершается.

конец

Разбор задачи по разработке алгоритма

Задание:

$$y = \frac{\sqrt{x^2 - x}}{\log_2 x + \log_x 2}$$

При
 $x_i = 1, 2, \dots, 100$

Решение:

