

## ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Слово «Алгоритм» происходит от algorithmi – латинского написания имени аль-Хорезми, под которым в средневековой Европе знали выдающегося математика с Хорезма (город в современном Узбекистане) Мухаммада ибн Мусу, жившего в 783-850 гг. Он сформулировал правила записи натуральных чисел с помощью арабских цифр и правила действий над ними столбиком.

В дальнейшем алгоритмом стали называть точное предписание, определяющее последовательность действий, обеспечивающую получение требуемого результата исходных данных. Алгоритм может быть предназначен для исполнения его человеком или автоматическим устройством. Создание алгоритма, пусть даже самого простого, – процесс творческий. Он доступен исключительно для живых существ, а долгое время считалось, что только человеку. Другое дело – реализация уже имеющегося алгоритма. Ее можно поручить субъекту или объекту, который не обязан вникать в суть дела, а может быть, и не способен его понять. Такой субъект или объект принято называть **формальным исполнителем**. В роли формального исполнителя может выступать и человек, но в первую очередь – разные автоматические устройства, и компьютер в том числе. Каждый алгоритм создается в расчете на вполне конкретного исполнителя. Те действия, которые может совершать исполнитель, называются его **допустимыми действиями**. Совокупность допустимых действий образует **систему команд исполнителя**. Алгоритм должен содержать только действия, допустимые для данного исполнителя. компьютера в том числе.

**Алгоритм** - точное и понятное предписание исполнителю осуществить последовательность действий, направленных на решение поставленной задачи.

**Исполнитель алгоритма** – это некоторая абстрактная или реальная (техническая, биологическая или биотехническая) система, способная выполнить действия, предусмотренные алгоритмом.

Исполнителя характеризуют:

- среда;
- элементарные действия;<sup>1</sup>
- система команд;
- отказы.

**Среда** (или обстановка) – это "место жительства" исполнителя. Например, для исполнителя Робота из школьного учебника среда – это бесконечное клеточное поле. Стены и окрашенные клетки тоже часть среды. А их расположение и положение самого Робота задают конкретное **состояние среды**.

**Система команд**. Каждый исполнитель может выполнять команды только из некоторого строго заданного списка – системы команд исполнителя. Для каждой команды должны быть заданы условия применимости (в каких состояниях среды может быть выполнена команда) и описаны результаты выполнения команды. К примеру, команда Робота "вверх" может быть выполнена, если выше Робота нет стены. Ее результат – переход Робота на одну клетку вверх.

После вызова команды исполнитель делает соответствующее элементарное действие.

**Отказы** исполнителя возникают, если команда вызывается при недопустимом для нее состоянии среды.

Обычно исполнитель ничего не знает о целях алгоритма. Он выполняет все полученные команды, не задавая вопросов "почему" и "зачем".

В информатике универсальным исполнителем алгоритмов является компьютер.

## **Свойства алгоритмов**

**Понятность** для исполнителя – то есть исполнитель алгоритма должен знать, как его выполнять.

**Дискретность** (прерывность, разрешение) – то есть алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов (этапов).

**Определенность** – то есть каждое правило алгоритма должно быть четким, однозначным и не оставлять места для неопределенности. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

**Результативность**. Это свойство состоит в том, что алгоритм должен приводить к решению задачи за конечное число шагов.

**Массовость**. Это значит, что алгоритм решения задачи разрабатывается в общем виде, т.е. применяемым для некоторого класса задач, различающихся только исходными данными. При этом исходные данные могут выбираться из некоторой области, называемой областью применимости алгоритма [1].

## **Формы записи алгоритмов**

На практике наиболее распространены следующие формы представления алгоритмов:

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

**Словесный способ** Запись алгоритмов описывает последовательные этапы обработки данных. Алгоритм представляется в произвольном изложении на естественном языке

**Например.** Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел.

Алгоритм может быть следующим:

1. задать два числа;

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

Описанный алгоритм применим к любому натуральному числу и должен приводить к решению поставленной задачи.

Убедитесь в этом самостоятельно, определив с помощью этого алгоритма самый общий делитель чисел 125 и 75.

Словесный способ не имеет распространения по следующим причинам:

- такие описания строго не формализуются;
- страдают многословием записей;
- допускают неоднозначность толкования.

**Графический способ** представление алгоритмов более компактно и наглядно по сравнению со словесным.

При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

Такое графическое представление называется схемой алгоритма или **блок-схемой**.

В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий.

В таблице 1.1 приведены наиболее часто используемые символы.

Таблица 1.1

**Графическое изображение алгоритма. Блочные символы**

Наименование	Обозначение	Функции
Процесс		Выполнение операции или группы операций, в результате которых меняется значение, форма представления или расположения данных
Вход-выход		Превращение данных в форму, пригодную для обработки (ввода) или отображения результатов обработки (вывода)
Решение		Выбор направления выполнения алгоритма в зависимости от некоторых условий
Модификация		Цикл. Выполнение действий заданное число раз
Обусловленный процесс		Использование ранее созданных и отдельно написанных программ (подпрограмм)
Документ		Вывод данных на бумажный носитель
Магнитный диск		Ввод-вывод данных, носителем которых служит магнитный диск
Пуск-останов		Начало, конец, прерывание процесса обработки данных
Соединитель		Указание связи между прерванными линиями, соединяющими блоки
Межстраничный соединитель		Указание связи между прерванными соединяющими линиями блоки, расположенные на разных листах
Комментарий		Связи между элементом схемы и пояснениями

Блок "**процесс**" применяется для обозначения действия или последовательности

действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок "**решения**" используется для обозначения переходов управления по условию. В каждом блоке "решения" должны быть указаны вопросы, условия или сравнения, которые он определяет.

Блок "**модификация**" используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок "**обусловленный процесс**" используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

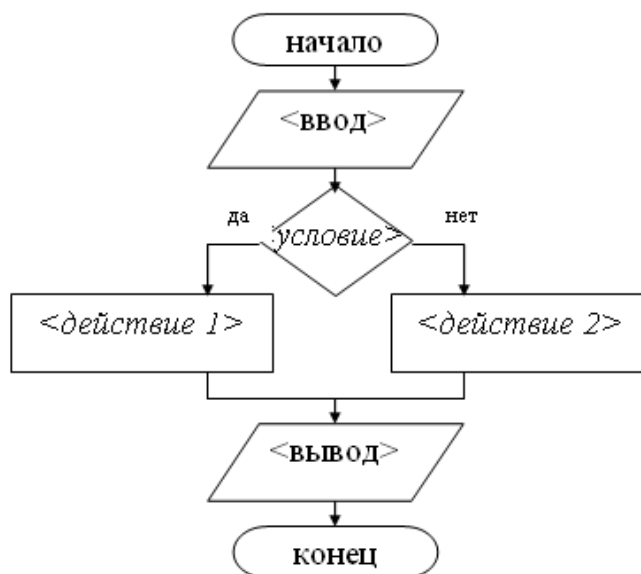


Рис. 1.1. Пример схемы алгоритма

**Псевдокод** представляет собой систему обозначений и правил, предназначенную для одинаковой записи алгоритмов. Он занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. Кроме того, в псевдокоде используются некоторые формальные конструкции и математическая символика, приближающая запись алгоритма к общепринятой математической записи. В псевдокоде не приняты строгие синтаксические правила для записи команд, свойственные формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя. Однако в псевдокоде обычно есть некоторые конструкции, свойственные формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, так же, как и в формальных языках, есть служебные слова, содержание которых определено раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются. Единого или формального

определения псевдокода не существует, поэтому возможны разные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций. Примером псевдокода является школьный алгоритмический язык (школьный АЯ).

## Запись алгоритмов на школьном алгоритмическом языке

Таблица 1.2

### Основные служебные слова

<b>алг</b> (алгоритм)	<b>сим</b> (символьный)	<b>дано</b>	<b>для</b>	<b>да</b>
<b>арг</b> (аргумент)	<b>лит</b> (буквенный)	<b>надо</b>	<b>вот</b>	<b>нет</b>
<b>рез</b> (результат)	<b>лог</b> (логический)	<b>если</b>	<b>к</b>	<b>при</b>
<b>нач</b> (начало)	<b>таб</b> (таблица)	<b>то</b>	<b>знач</b>	<b>выбор</b>
<b>кон</b> (конец)	<b>нц</b> (начало цикла)	<b>иначе</b>	<b>и</b>	<b>ввод</b>
<b>цел</b> (целый)	<b>кц</b> (конец цикла)	<b>все</b>	<b>или</b>	<b>вывод</b>
<b>вещ</b> (действительный)	<b>длин</b> (длина)	<b>пока</b>	<b>не</b>	<b>утв</b>

#### Общий вид алгоритма:

**алг** название алгоритма (аргументы и результаты)

**дано** условия применимости алгоритма

**надо** цель выполнения алгоритма

**нач** описание промежуточных величин

| последовательность команд (тело алгоритма)

**кон**

Часть алгоритма от слова **алг** к слову **нач** называется заголовком, а часть, которая находится между словами **нач** и **кон** - телом алгоритма.

В предложении **алг** после названия алгоритма в круглых скобках указываются характеристики (**арг**, **рез**) и тип значения (**цел**, **вещ**, **сим**, **лит** или **лог**) всех входных (аргументы) и выходных (результаты) переменных. При описании массивов (таблиц) используется служебное слово **таб**, дополненное граничными парами по каждому индексу элементов массива.

Примеры предложений **алг**:

1. **алг** Объем и площадь цилиндра (**арг вещ** R, H, **рез вещ** V, S)

2. **алг** Корни КвУр (**арг вещ** a, b, c, **рез вещ** x1, x2, **рез лит** t)

3. **алг** Исключить элемент (**арг цел** N, **арг рез вещ таб** A [1: N])

4. **алг** Диагональ (**арг цел** N, **арг цел таб** A [1: N, 1: N], **рез лит** Otvet)

Предложения **дано** и **надо** не обязательны. В них рекомендуется записывать утверждения, описывающие состояние среды исполнителя алгоритма, например:

**алг** Замена (**арг лит** Str1, Str2, **рез лит** Text)

**дано** | длины подстрок Str1 и Str2 совпадают

**надо** | повсюду в строке Text подстрока Str1 заменена Str2

**алг** Число максимумов (**арг цел** N, **арг вещ таб** A [1:N], **рез цел** K)

**дано** |  $N > 0$

**надо** |  $K$  – количество максимальных элементов в таблице  $A$

**алг** Основание (**арг вещь**  $R_1, R_2$ , **арг цел**  $N$ , **рез вещь**  $R$ )

**дано** |  $N > 5, R_1 > 0, R_2 > 0$

**надо** |  $R$  – основание схемы

Здесь в предложениях **дано** и **надо** после знака "|" записаны комментарии. Комментарии можно помещать в конце любой строчки. Они не обрабатываются транслятором, но значительно упрощают понимание метода.

### Команды школьного АЯ

**Оператор присваивания.** Служит для вычисления выражений и присваивания их значений переменным. Общий вид:  $A := B$ , где знак " $:=$ " означает команду заменить прежнее значение переменной, стоящей в левой части, на вычисленное значение выражения, стоящего в правой части.

Например,  $a := (b+c) * \sin(\pi/4)$ ;  $i := i+1$ .

Для ввода и вывода данных используют команды:

- **ввод** имена переменных
- **вывод** имена переменных, выражения, тексты.

Для **разветвления** применяют команды **если** и **выбор**,

Для **организации циклов** - команды **для** и **пока**.

### **Пример записи алгоритма на школьном АЯ**

**алг** Сумма квадратов (**арг цел**  $n$ , **рез цел**  $S$ )

**дано** |  $n > 0$

**надо** |  $S = 1*1 + 2*2 + 3*3 + \dots + n*n$

**нач** цел  $i$

**ввод**  $n$ ;  $S := 0$       7

**нц для**  $i$  **от** 1 **до**  $n$

$S := S + i*i$

**кц**

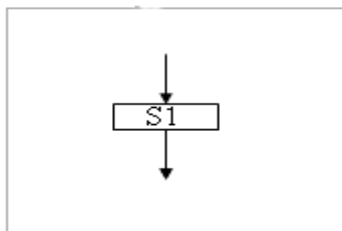
**вывод** " $S =$ ",  $S$

**кон**

## Основные алгоритмические конструкции

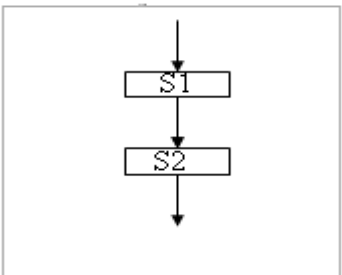
### Изображение

### Описание



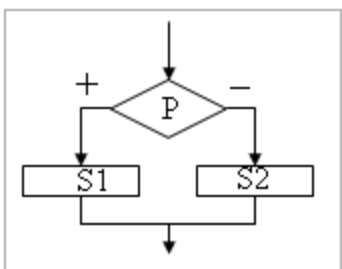
Данный блок имеет *один вход* и *один выход*. Из простых команд и проверки условий образуются составные команды, имеющие более сложную структуру и тоже *один вход* и *один выход*.

Структурный подход к разработке алгоритмов определяет использование только базовых алгоритмических структур (конструкций): следование, ветвление, повторение, которые должны быть оформлены стандартным образом.

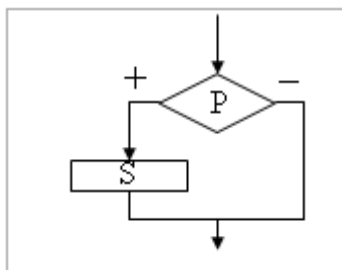


Рассмотрим основные структуры алгоритма.

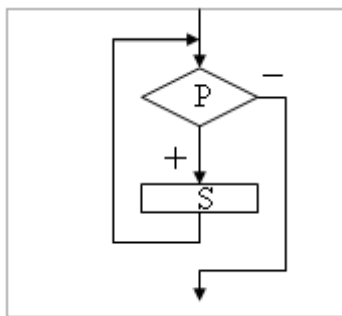
Команда *следования* состоит только из простых команд. На рисунке простые команды имеют условное обозначение  $S1$  и  $S2$ . Из команд следования образуются линейные алгоритмы. Примером линейного алгоритма будет нахождение суммы двух чисел, введенных с клавиатуры.



Команда *ветвления* - это составная команда алгоритма, в которой в зависимости от условия  $P$  выполняется или одно  $S1$ , или другое  $S2$  действие. Из команд следования и команд ветвления составляются разветвляющиеся алгоритмы (алгоритмы ветвления). Примером разветвляющегося алгоритма будет нахождение большего из двух чисел, введенных с клавиатуры.



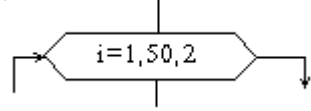
Команда ветвления может быть полной и неполной формы. Неполная форма команды ветвления используется тогда, когда необходимо выполнять действие  $S$  только в случае соблюдения условия  $P$ . Если условие  $P$  не соблюдается, то команда ветвления завершает свою работу без выполнения действия. Примером команды ветвления неполной формы будет уменьшение в два раза только четного числа.



Команда *повторения* - это составная команда алгоритма, в которой в зависимости от условия  $P$  возможно многократное выполнение действия  $S$ . Из команд следования и команд повторения составляются циклические алгоритмы (алгоритмы повторения). На рисунке представлена команда повторения с предусловием. Называется она так потому, что вначале проверяется условие, а уже затем выполняется действие. Причем действие выполняется, пока условие соблюдается. Пример циклического алгоритма может быть следующий. Пока с клавиатуры вводятся положительные числа, алгоритм выполняет нахождение их суммы.



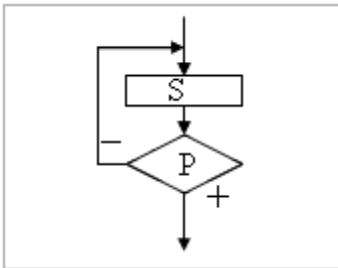
Команда повторения с предусловием не является единственно возможной. Разновидностью команды повторения с предусловием является команда повторения с параметром. Она используется тогда, когда известно количество повторений действия. В блок-схеме команды повторения с параметром условие записывается не в ромбе, а в шестиугольнике.



Примером циклического алгоритма с параметром будет нахождение суммы нечетных чисел от 1 до 50.

В команде повторения с постусловием вначале выполняется действие *S* и лишь затем, проверяется условие *P*. Причем действие повторяется до тех пор, пока условие не соблюдается.

Примером команды повторения с постусловием будет уменьшение положительного числа до тех пор, пока оно неотрицательное. Как только число становится отрицательным, команда повторения заканчивает свою работу.



**С помощью соединения только этих элементарных конструкций (последовательно или вложением) можно "собрать" алгоритм любой степени сложности.**

Способы соединения базовых структур алгоритма представлены на рисунке 1.2.

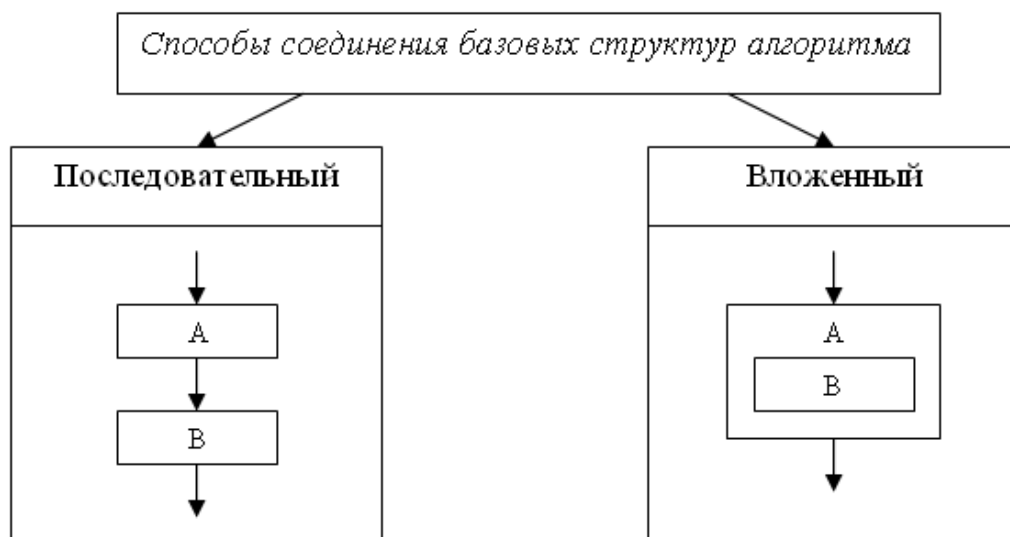


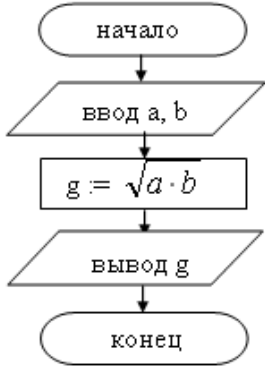
Рис. 1.2. Способы соединения базовых структур алгоритма

Каждая указанная конструкция может быть без изменений в структуре реализована на любом языке программирования. Поэтому необходимо грамотно составить алгоритм с помощью блок-схемы, а уже затем, зная, как записываются команды на конкретном языке программирования, набрать программу на компьютере и получить результат, запустив ее на исполнение.

### 1.6.1. Линейный алгоритм

Приведем пример записи алгоритма в виде блок-схемы, псевдокодов и на языке Паскаль, C++.

$$g = \sqrt{ab},$$

Блок-схема	Псевдокоды	Паскаль
 <pre> graph TD     Start([начало]) --&gt; Input[/ввод a, b/]     Input --&gt; Process[g := √(a · b)]     Process --&gt; Output[/вывод g/]     Output --&gt; End([конец])         </pre>	<p><b><u>алг</u></b> среднее геометрическое</p> <p><b><u>вещ</u></b> a, b, g</p> <p><b><u>нач</u></b></p> <p><b><u>ввод</u></b> a, b</p> <p>g := (a * b) ^ (1/2)</p> <p><b><u>вывод</u></b> g</p> <p><b><u>кон</u></b></p>	<pre> <b>program</b> Srednee_geometr;   <b>var</b> a, b, g: real; <b>begin</b>     <b>readln</b> (a, b);     s := sqrt(a * b);     <b>writeln</b> (g) <b>end.</b>         </pre>

#### C++

```

void main()
{
    double a, b, g;
    cin >> a >> b;
    g = sqrt(a * b);
    cout << g;
}
        
```

## 1.6.2. Разветвляющийся алгоритм

Приведем пример записи разветвляющегося алгоритма для нахождения наибольшего из двух чисел.

Блок – схема	Псевдокоды	Pascal
<pre> graph TD     Start([начало]) --&gt; Input[/Ввод a, b/]     Input --&gt; Decision{a &gt; b}     Decision -- да --&gt; AssignA[max:=a]     Decision -- нет --&gt; AssignB[max:=b]     AssignA --&gt; Output[/Вывод max/]     AssignB --&gt; Output     Output --&gt; End([конец])         </pre>	<p><b><u>ал</u></b> большее из чисел</p> <p><b><u>вещ</u></b> a, b, max</p> <p><b><u>нач</u></b></p> <p>    <b><u>ввод</u></b> a, b</p> <p>    <b><u>если</u></b> a&gt;b</p> <p>        <b><u>то</u></b> max:=a</p> <p>    <b><u>иначе</u></b> max:=b</p> <p>    <b><u>все</u></b></p> <p>    <b><u>вывод</u></b> max</p> <p><b><u>кон</u></b></p>	<pre> program ostatok; var a, b, max: real; begin     readln (a, b);     if a&gt;b         then max:=a         else max:=b;     writeln (max) end.         </pre>

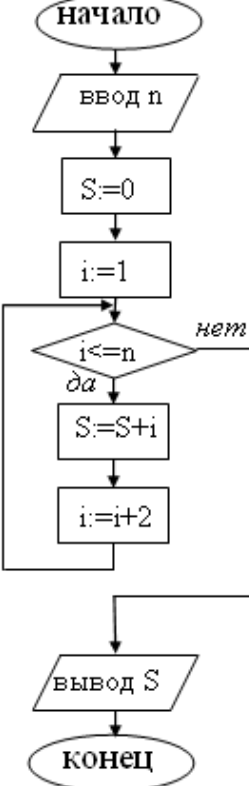
C++

```

void main()
{
    double a, b, max;
    cin >> a >> b;
    if (a > b) max = a;
    else max = b;
    cout << max;
}
    
```

### 1.6.3. Циклический алгоритм

Рассмотрим алгоритм нахождения суммы первых натуральных нечетных чисел до  $n$ . Представим запись алгоритма тремя способами: в виде блок-схемы, школьного алгоритмического языка и на языке программирования Pascal, C++.

Блок – схема	Псевдокоды	Pascal
 <pre> graph TD     Start([Начало]) --&gt; Input[/Ввод n/]     Input --&gt; S0[S:=0]     S0 --&gt; i1[i:=1]     i1 --&gt; Decision{i &lt;= n}     Decision -- да --&gt; Sum[S:=S+i]     Sum --&gt; Inc[i:=i+2]     Inc --&gt; Decision     Decision -- нет --&gt; Output[/Вывод S/]     Output --&gt; End([Конец])         </pre>	<p><u>алг</u> сумма нечетных чисел</p> <p><u>нач</u></p> <p><u>ввод</u> n</p> <p>S:=0</p> <p>i:=1</p> <p><u>нц пока</u> i&lt;=n</p> <p>S:=S+i</p> <p>i:=i+2</p> <p><u>кц</u></p> <p><u>вывод</u> S</p> <p><u>кон</u></p>	<pre> program summa_nech; var i, n, S: integer; begin   readln (n);   S:=0;   i:=1;   while i&lt;=n do     begin       S:=S+i;       i:=i+2;     end;   writeln (S) end.         </pre>

C++

```

void main()
{
  int i, n, S;
  cin >> n;
  S = 0;
  i = 1;
  while (i <= n)
  {
    S += i;
    i += 2;
  }
  cout << S;
}
        
```

Пунктирные стрелки в таблице отражают последовательность выполнения технологической цепочки. После записи алгоритма в виде блок-схемы каждая команда переводится на школьный алгоритмический язык, а уже затем на язык программирования.

Запишем алгоритм вычисления суммы первых n натуральных чисел. Для этого воспользуемся циклом с параметром, поскольку заранее известно сколько раз будет выполняться команда нахождения суммы. Во всех звеньях цепочки поменяем цикл "пока" на цикл "для" и приведем пример перевода алгоритма с языка блок-схем на школьный алгоритмический язык и на язык программирования Pascal, C++.

Блок – схема	Псевдокоды	Pascal
<pre> graph TD     Start([начало]) --&gt; Input[/Ввод n/]     Input --&gt; S0[S:=0]     S0 --&gt; LoopStart{i:=1, n}     LoopStart -- да --&gt; Sum[S:=S+i]     Sum --&gt; LoopStart     LoopStart -- нет --&gt; Output[/Вывод S/]     Output --&gt; End([конец]) </pre>	<pre> алг сумма чисел нач     ввод n     S:=0     <u>нц</u> для i от 1 до n         S:=S+i     <u>кц</u>     вывод S кон </pre>	<pre> program summa; var i, n, S: integer; begin     readln (n);     S:=0;     for i:=1 to n do         S:=S+i;     writeln (S) end. </pre>

12

### C++

```

void main()
{
    int i, n, S;
    cin >> n;
    S = 0;
    for (i = 1; i <= n; i++)
        S = S + i;
    cout << S;
}

```

Рассмотрим нахождение количества натуральных чисел, сумма которых не больше заданной. Для этого воспользуемся командой повторения с постусловием.

Блок – схема	Псевдокоды	Pascal
<pre> graph TD     Start([Начало]) --&gt; Input[/Ввод Q/]     Input --&gt; S0[S:=0]     S0 --&gt; i1[i:=1]     i1 --&gt; Splus[S:=S+i]     Splus --&gt; iplus[i:=i+1]     iplus --&gt; Decision{S&gt;Q}     Decision -- да --&gt; Output[/Вывод (i-2)/]     Decision -- нет --&gt; Splus     Output --&gt; End([Конец])         </pre>	<p><b><u>алг</u></b> количество чисел</p> <p><b><u>нач цел</u></b> i, S, Q</p> <p><b><u>ввод</u></b> Q</p> <p>S:=0</p> <p>i:=1</p> <p><b><u>делать</u></b></p> <p>S:=S+i</p> <p>i:=i+1</p> <p><b><u>до</u></b> S&gt;Q</p> <p><b><u>вывод</u></b> (i-2)</p> <p><b><u>кон</u></b></p>	<p><b>program</b> kolichество;</p> <p>var i, S, Q: integer;</p> <p><b>begin</b></p> <p>  <b>readln</b> (Q);</p> <p>  S:=0;</p> <p>  i:=1;</p> <p>  <b>repeat</b></p> <p>    S:=S+i;</p> <p>    i:=i+1</p> <p>  <b>until</b> S&gt;Q;</p> <p>  <b>writeln</b> (i-2)</p> <p><b>end.</b></p>

```

C++
void main()
{
    int i, S, Q;
    cin >> Q;
    S = 0;
    i = 1;
    do
        S += i;
        i++;
    while (S <= Q);
    cout << i - 2;
}

```

#### 1.6.4. Вспомогательный (подчиненный) алгоритм

*Вспомогательный* (подчиненный) алгоритм (процедура) – алгоритм, ранее разработанный и целиком используемый при алгоритмизации конкретной задачи. В некоторых случаях при наличии одинаковых последовательностей указаний (команд) для различных данных с целью сокращения записи также выделяют вспомогательный алгоритм.

## Отличие программного способа записи алгоритмов от других

При записи алгоритма в словесной форме, в виде блок-схемы или на псевдокоде допускается определенная вольность при изображении команд. Вместе с тем такая запись является точной настолько, что позволяет человеку понять суть дела и выполнить алгоритм.

Однако на практике в качестве исполнителей алгоритмов используются специальные автоматы – компьютеры. Поэтому алгоритм, предназначенный для выполнения на компьютере, должен быть записан «понятным» для него языком. И здесь на первый план выдвигается необходимость точной записи команд, которая не оставляет места для произвольного толкования их исполнителем.

Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке – программой для компьютера.

### Уровень языка программирования

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Любой алгоритм представляет собой последовательность действий, выполнив которые можно за конечное число шагов перейти от исходных данных к результату. В зависимости от степени детализации обычно определяется уровень языка программирования – чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие **уровни языков программирования**:

- машинные;
- машинно-ориентированные ( а с с е м б л е р ы );
- машинно-независимые (языки высокого уровня).

**Машинные языки и машинно-ориентированные языки** - это языки **низкого уровня**, требующие указаний<sup>15</sup> на мельчайшие детали процесса обработки данных.

Языки высокого уровня имитируют естественные языки, используя некоторые слова разговорной речи и общепринятые математические символы. Эти языки более удобны для человека.

**Языки высокого уровня делятся на:**

- **алгоритмические** (Basic, Pascal, C и др.), предназначенные для однозначного описания алгоритмов;
- **логические** (Prolog, Lisp и др.), ориентированные не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания.
- **объектно-ориентированные** (Object Pascal, C++, Java и др.), в основе которых лежит понятие объекта, совмещающего в себе данные и действия над ними. Программа на объектно-ориентированном языке, решая некоторую задачу, по существу описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы

взаимодействующих объектов более естественно, чем в форме взаимодействующих процедур.

### **Преимущества и недостатки машинных языков**

Каждый компьютер имеет свой машинный язык, то есть свою совокупность машинных команд, отличающуюся количеством адресов в команде, назначением информации, задаваемой в адресах, набором операций, которые может выполнить машина и т.д.

При программировании на машинном языке программист может держать под контролем каждую команду и каждую ячейку памяти, использовать все возможности имеющихся машинных операций.

Но процесс написания программы на машинном языке очень трудоемок и утомителен. Программа выходит громоздкой, ее трудно отлаживать, изменять и развивать.

Поэтому в случае, когда нужно иметь эффективную программу, в максимальной степени учитывающую специфику конкретного компьютера, вместо машинных языков используют близкие к ним машинно-ориентированные языки (ассемблеры).

### **Язык ассемблера**

Язык ассемблера – это система обозначений, которая используется для представления в понятной форме программ, написанных в машинном коде.

Она позволяет программисту пользоваться текстовыми мнемоническими (т.е. легко запоминающимися человеком) кодами, по своему усмотрению присваивать символические имена регистров компьютера и памяти, а также задавать удобные для себя способы адресации. Кроме того, она позволяет использовать различные системы счисления (например, десятичную или шестнадцатеричную) для представления числовых констант, использовать в программе комментарии и т.п.

Перевод программы с языка ассемблера на машинный язык осуществляется специальной программой, которая также называется ассемблером и является по сути простейшим транслятором [2].

### **Преимущества алгоритмических языков перед машинными**

Основные преимущества таковы:

- алфавит алгоритмического языка значительно шире алфавита машинного языка, что существенно повышает наглядность текста программы;
- набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулировки алгоритмов решения задач определенного класса;
- формат предложений достаточно гибок и удобен для использования, что позволяет с помощью одного предложения задать достаточно содержательный этап обработки данных;



- необходимые операции задаются с помощью общепринятых математических обозначений;
- данным в алгоритмических языках присваиваются индивидуальные имена, избранные программистом;
- в языке может быть предусмотрен более широкий набор типов данных по сравнению с набором машинных типов данных.

Таким образом, *алгоритмические языки в значительной степени являются машинно-независимыми. Они упрощают работу программиста и увеличивают надежность создаваемых программ.*

### **Компоненты, образующие алгоритмический язык**

Алгоритмический язык (как и любой другой язык) образуют три его составляющие: **алфавит, синтаксис и семантика.**

**Алфавит** - это фиксированный для данного языка набор основных символов, то есть "букв алфавита", из которых должен состоять любой текст на этом языке – никакие другие символы в тексте не допускаются.

**Синтаксис** - это правила построения фраз, позволяющие определить, правильно или неправильно написана та или иная фраза. Точнее говоря, синтаксис языка представляет собой набор правил, устанавливающих какие комбинации символов являются содержательными предложениями на этом языке.

**Семантика** определяет содержательное значение предложений языка. Будучи системой правил толкования отдельных языковых конструкций, семантика устанавливает, какие последовательности действий описываются теми или иными фразами языка и, в конечном счете, какой алгоритм определен по этому тексту на алгоритмическом языке.

### **Понятия, используемые в алгоритмических языках**

Каждое понятие алгоритмического языка подразумевает некоторую синтаксическую единицу (конструкцию) и определяемые ею свойства программных объектов или процесса обработки данных.

**Понятие языка** определяется во взаимодействии синтаксических и семантических правил. Синтаксические правила показывают, как появляется данное понятие из других понятий и букв алфавита, а семантические правила определяют свойства данного понятия.

**Основными понятиями в алгоритмических языках обычно являются следующие:**

**Имена** (идентификаторы) - используются для обозначения объектов программы (переменных, массивов, функций и др.).

**Операции.** Типы операций:

- **арифметические** операции +, -, \*, / и др.;
- **логические** операции и, или, не;

- **операции отношения**  $<$ ,  $>$ ,  $<=$ ,  $>=$ ,  $=$ ,  $<>$ ;
- **операция сцепки** (иначе "присоединение", "конкатенации") символьных значений друг с другом с образованием одной длины строки; изображается знаком "+".

**Данные** – величины, обрабатываемые программой. Есть три основных вида данных: константы, переменные и массивы.

• **Константы** – это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения.

Примеры констант:

- **числовые** 7.5, 12;
- **логические** да (истина), нет (ложь);
- **символьные** "А", "+";
- **буквенные** "abcde", "информатика", "" (пустая строка).

• **Переменные** обозначаются именами и могут изменять свои значения в ходе выполнения программы. **Переменные** бывают целые, вещественные, логические, символьные и буквенные.

• **Массивы** – последовательности однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним, в случае одномерного массива, или несколькими, если массив многомерен). Иногда массивы называются таблицами.

**Выражения** - назначаются для выполнения необходимых вычислений, состоят из констант, переменных, указателей функций (например  $\exp(x)$ ), объединенных знаками операций.

Выражения записываются в виде линейных последовательностей символов (без подстрочных и надстрочных символов, "многоэтажных" дробей и т.д.), позволяющих вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры.

Различают выражения арифметические, логические и строчные.

• **Арифметические выражения служат для определения одного числового значения.** Например,  $(1+\sin(x))/2$ . Значение этого выражения при  $x = 0$  равно 0.5, а при  $x = \pi/2$  – единицы.

• **Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться.** Таким образом, логическое выражение может принимать только два значения – "истина" или "ложь" (да или нет).

Рассмотрим, как пример, логическое выражение  $x^2+y^2 < r^2$ , определяющее принадлежность точки с координатами (x, y) внутренней области окружности радиусом r с центром в начале координат. При  $x = 1$ ,  $y = 1$ ,  $r = 2$  значение этого выражения – "истина", а при  $x = 2$ ,  $y = 2$ ,  $r = 1$  – "ложь".

• **Значение строчных (буквенных) выражений – текст.** В них могут входить буквенные константы, буквенные переменные и буквенные функции, разделенные знаком операции сцепки. Например, A+B означает присоединение строки B к концу строки A. Если A = "куст", а B = "зеленый", то значение выражения A + B

есть "куст зеленый".

**Операторы** (команды). Оператор – это наиболее большое и содержательное понятие языка: каждый оператор является законченной фразой языка и означает некий полностью законченный этап обработки данных. В состав операторов входят: ключевые слова, данные, выражения и т.д.

Операторы делятся на выполняемые и невыполняемые. Невыполняемые операторы предназначены для описания данных и структуры программы, а выполняемые – для выполнения различных действий (например, оператор присваивания, операторы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и др.).

### **Стандартные функции**

При решении разных задач с помощью компьютера бывает необходимо вычислить логарифм или модуль числа, синус угла и т.д. Вычисления часто используемых функций осуществляются с помощью подпрограмм, так называемые стандартные функции, заранее запрограммированные и встроенные в транслятор языка.

Таблица 1.8

**Таблица стандартных функций школьного алгоритмического языка**

Название и математическое обозначение функции	Указатель функции	
	$ x $	abs(x)
Корень квадратный	$\sqrt{x}$	sqrt(x)
Натуральный логарифм	$\ln x$	ln(x)
Десятичный логарифм	$\lg x$	lg(x)
Экспонента (степень числа $e^{2.72}$ )	$e^x$	exp(x)
Знак числа x (-1, если $x < 0$ ; 0, если $x = 0$ ; 1, если $x > 0$ )	sign x	sign(x)
Целая часть x (т.е. максимальное целое число, не превышающее x)		int(x)
Минимум из чисел x и y		min(x,y)
Максимум из чисел x и y		max(x,y)
Доля от деления целого x на целое y		div(x,y)
Остаток от деления целого x на целое y		mod(x,y)
Случайное число в диапазоне от 0 до x-1		rnd(x)
Синус (угол в радианах)	$\sin x$	sin(x)
Косинус (угол в радианах)	$\cos x$	cos(x)
Тангенс (угол в радианах)	$\operatorname{tg} x$	tg(x)
Котангенс (угол в радианах)	$\operatorname{ctg} x$	ctg(x)
Арксинус (главное значение в радианах)	$\arcsin x$	arcsin(x)

Арккосинус (главное значение в радианах)	arccos x	arccos(x)
Арктангенс (главное значение в радианах)	arctg x	arctg(x)
Арккотангенс (главное значение в радианах)	arcctg x	arcctg(x)

Каждый язык программирования имеет набор стандартных функций.

В качестве аргументов функций можно использовать константы, переменные и выражения.

Например:

$\sin(3.05)$      $\sin(x)$      $\sin(2*y+t/2)$      $\sin((\exp(x)+1)**2)$   
 $\min(a, 5)$      $\min(a, b)$      $\min(a+b, a*b)$      $\min(\min(a,b), \min(c,d))$

### Запись арифметических выражений

Арифметические выражения записываются по следующим правилам:

- Нельзя опускать символ умножения меж сомножителями и ставить рядом два знака операций.
- Индексы частей массивов записываются в квадратных (школьный АЯ, Pascal, C) либо круглых (Basic) скобках.
- Для обозначения переменных употребляются буквы латинского алфавита.
- Операции выполняются в порядке старшинства: сначала вычисление функций, затем возведение в степень, затем умножение и деление и в последнюю очередь – сложение и вычитание.
- Операции одного старшинства выполняются слева направо.

Таблица 1.9

#### Примеры записи арифметических выражений

Математическая запись	Запись на ПЯ
$\frac{xy}{z}$	$x*y/z$
$\frac{x}{yz}$	$x/(y*z)$ или $x/y/z$
$\frac{a^3 + b^3}{bc}$	$(a**3+b**3)/(b*c)$
$(a_{i+1} + b_{i-1})^2 xy$	$(a[i+1]+b[i-1])**2*x*y$
$0.49e^{a^3-b^3} + \ln^3 \cos a^2$	$0.49*\exp(a*a-b*b)+\ln(\cos(a*a))**3$

Самые типовые ошибки в записи выражений:

$5x+1$     Пропущенный знак умножения между 5 и x  
 $a+\sin x ((a+b)/c**3$     Аргумент x функции  $\sin x$  не помещен в скобки  
 $((a+b)/c**3$     Недостает закрывающей скобки

## Запись логических выражений

В записи логических выражений кроме арифметических операций сложения, вычитания, умножения, деления и подъема к степени используются операции отношения  $<$  (меньше),  $\leq$  (меньше или равно),  $>$  (больше),  $\geq$  (больше или равно),  $=$  (равно),  $\neq$  (не равно), а также логические операции и, или, не.

Таблица 1.10

### Примеры записи логических выражений

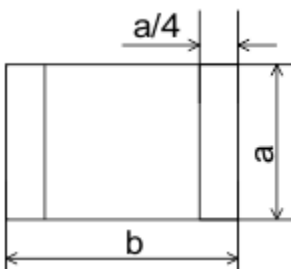
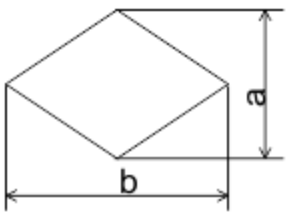
Условие	Запись на школьном алгоритмическом языке
Дробная часть действительного числа $a$ равна нулю	$\text{int}(a) = 0$
Целое число $a$ – четное	$\text{mod}(a, 2) = 0$
Целое число $a$ – нечетное	$\text{mod}(a, 2) = 1$
Целое число $k$ кратно семи	$\text{mod}(a, 7) = 0$
Каждое из чисел $a, b$ положительно	$(a > 0) \text{ и } (b > 0)$
Только одно из чисел $a, b$ положительно	$((a > 0) \text{ и } (b \leq 0)) \text{ или } ((a \leq 0) \text{ и } (b > 0))$
Хотя бы одно из чисел $a, b, c$ является отрицательным	$(a < 0) \text{ или } (b < 0) \text{ или } (c < 0)$
Число $x$ удовлетворяет условию $a < x < b$	$(x > a) \text{ и } (x < b)$
Число $x$ имеет значение в промежутке $[1, 3]$	$(x \geq 1) \text{ и } (x \leq 3)$
Целые числа $a$ и $b$ имеют одинаковую четность	$((\text{mod}(a, 2) = 0) \text{ и } (\text{mod}(b, 2) = 0))$ <b>или</b> $((\text{mod}(a, 2) = 1) \text{ и } (\text{mod}(b, 2) = 1))$
Точка с координатами $(x, y)$ лежит в круге радиуса $r$ с центром в точке $(a, b)$	$(x+a)**2 + (y+b)**2 < r*r$
Уравнения $ax^2 + bx + c = 0$ нет действительных корней	$b*b - 4*a*c < 0$
Точка $(x, y)$ принадлежит первому или третьему квадранту	$((x > 0) \text{ и } (y > 0)) \text{ или } ((x < 0) \text{ и } (y < 0))$
Целые числа $a$ и $b$ являются взаимно противоположными	$a = -b$
Целые числа $a$ и $b$ являются взаимно обратными	$a*b = 1$
Число $a$ не менее среднего арифметического чисел $b, c, d$	$a \geq (b+c+d)/3$
Хотя бы одна из логических переменных $F1$ и $F2$ имеет значение да	$F1 \text{ или } F2$
Обе логические переменные $F1$ и $F2$ имеют значение да	$F1 \text{ и } F2$
Обе логические переменные $F1$ и $F2$ имеют значение нет	$\text{не } F1 \text{ и не } F2$
Логическая переменная $F1$ имеет значение да, а логическая переменная $F2$ имеет значение нет	$F1 \text{ и не } F2$

## Гост на описание блок-схем

Основные блоки, используемые для составления схем алгоритмов, представлены в нормативных документах ЕСПД, главным образом это

- ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначения условные графические
- ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные Обозначения и правила выполнения

### **Основные блоки для составления алгоритмов**

Название	Обозначение	Описание
Терминатор		Начало, конец, прерывание процесса обработки данных или выполнения программы
Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных
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)
Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий Блок решения имеет 1 вход и по крайней мере 2 выхода

Название	Обозначение	Описание
Границы цикла	<p>Начало цикла</p>  <p>Конец цикла</p> 	<p>Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один идентификатор.</p> <p>Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.</p>
Подготовка		<p>Выполнение операций, меняющих команды или группу команд, с целью воздействия на некоторую последующую функцию (установка переключателя, модификация регистра, инициализация программы)</p>
Комментарий		<p>Пояснение к элементу схемы (или линии связи)</p>
Соединитель		<p>При большой насыщенности схемы отдельные линии потока между удаленными символами допускается обрывать. При этом в конце (начале) обрыва должен быть помещен символ «Соединитель». Внутри блока соединителя указывается имя уникального идентификатора.</p>

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

Основным направлением потока в схемах алгоритмов принято направление сверху-вниз, слева-направо. Если линии потока идут в основном направлении и не имеют изломов, стрелками их можно не обозначать. В остальных случаях направление линии потока обозначать стрелкой обязательно.

Записи внутри символа должны быть представлены так, чтобы их можно было читать слева направо и сверху вниз, независимо от направления потока.

В схеме символу может быть присвоен идентификатор, который должен

помещаться слева над символом.

Допускается краткая информация о символе (описание, уточнение или другие перекрестные ссылки для более полного понимания функции данной части схемы). Описание символа должно помещаться справа над символом.

В случае необходимости слияния линий потока место слияния должно быть обозначено точкой или символом 0.

