

Основы алгоритмизации и программирования

1. Алгоритмизация

Для успешного использования ЭВМ в своей профессиональной деятельности пользователь должен уметь формулировать задачи, разрабатывать алгоритмы их решения, записывать алгоритмы на языке, понятном ЭВМ.

Этапы подготовки и решения реальных задач приведены на рис. 1.

В курсе информатики детально рассматриваются 4, 5 и 6 этапы решения задач, так как они непосредственно связаны с использованием ЭВМ.



Рис. 1. Этапы решения задач на ЭВМ

1.1. Алгоритмы и способы их описания

Алгоритм — система правил, рецептов, инструкций. Алгоритм — точный порядок действий, определяющий процесс, ведущий от исходных данных к искомому результату и обладающий следующими свойствами:

- определенностью, т.е. общепонятностью и точностью;
- массовостью, т.е. возможностью использования различных данных при решении однотипных задач;
- результативностью, т.е. направленностью на получение искомого результата;
- дискретностью, при выполнении разбивается на конечную последовательность действий или шагов;
- конечностью, т.е. должен выполняться за конечное время.

Для представления алгоритмов используются несколько способов:

- словесный (описание на естественном человеческом языке);
- графический (на языке блок-схем);
- с помощью символов специального языка проектирования программ-псевдокодов;
- с использованием НПО-диаграмм;
- с использованием таблиц решений;
- с помощью схемы Насси — Шнейдермана;
- с помощью одного из алгоритмических языков программирования.

Типовые этапы разработки алгоритмов:

описание общего замысла алгоритма;

формализация задачи;

разработка обобщенной схемы алгоритма;

разработка отдельных блоков алгоритма;

стыковка блоков;

определение возможности использования стандартных блоков;

разработка блоков логического контроля;

оптимизация схемы алгоритма;

уточнение параметров;

оценка машинного ресурса.

Hierarchical input process output (HIPO) — технология проектирования и документирования.

Фирма IBM создала методологию диаграмм IPO (вход-обработка-выход) в 1970-ых. Диаграммы IPO (или спецификации интерфейсов) являются основными в технологии.

Согласно технологии HIPO — используется некоторый формализованный и регламентированный подход к проектированию (документированию).

В IPO диаграммах выделены 3 колонки:

в левой записывается входная информация (та, что подается на вход процесса);

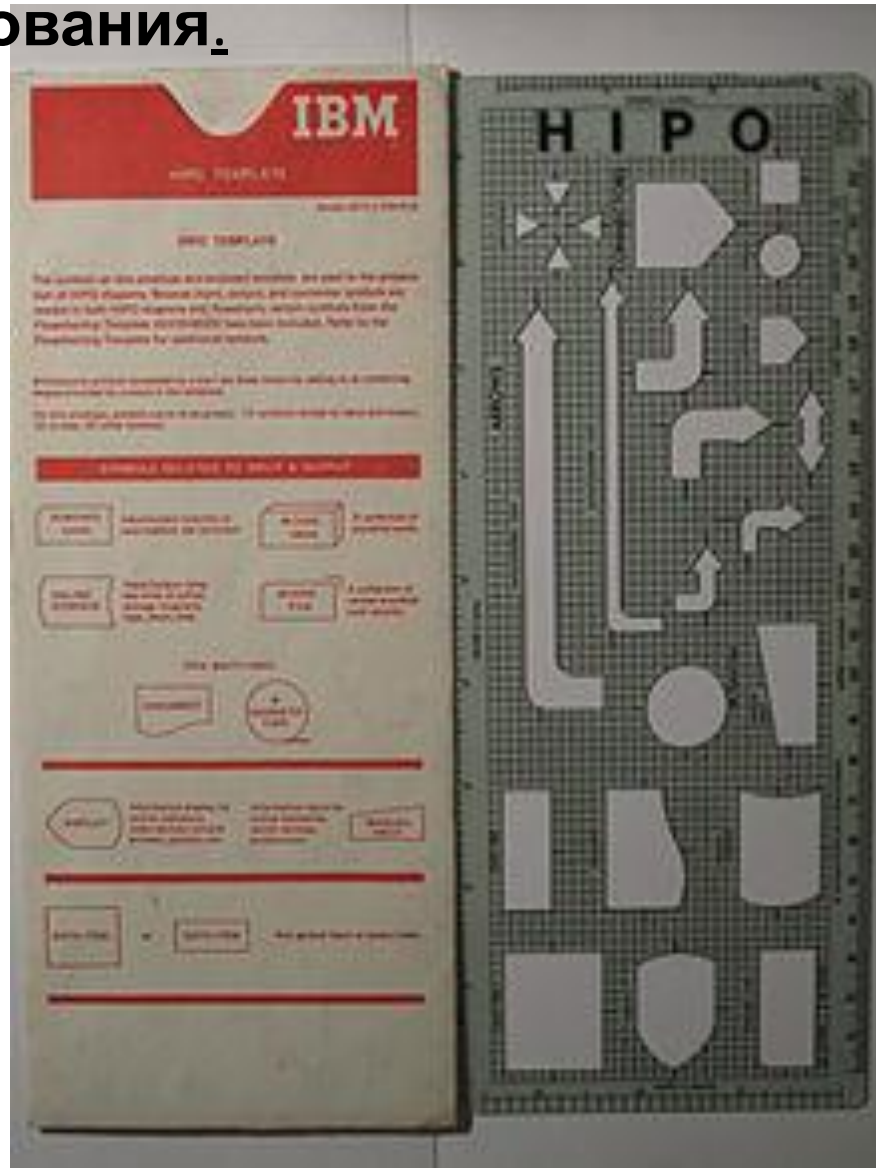
в средней описан процесс (алгоритм);

в правой — выходная информация (процесса).

Язык заполнения IPO диаграмм не оговаривается и может быть любым. Согласно HIPO технологии, процесс проектирования системы заканчивается только после окончания заполнения всех IPO диаграмм и увязки их друг с другом.

Все IPO диаграммы имеют строго формализованную систему ссылок,

HIPO - технология проектирования и документирования.



IPO - диаграммы

| Вход | Обработка | Выход |
|--------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| А,В,С Действительные неотрицательные значения | Если А,В,С – стороны треугольника, вычислить S – площадь треугольника с длинами сторон А,В,С, в противном случае вывести соответствующее сообщение М. | S – действительное неотрицательное значение М – сообщение |

Таблица принятия решений (таблица решений) — способ компактного представления модели со сложной **логикой**. Аналогично условным операторам в языках программирования, они устанавливают связь между условиями и действиями. Но, в отличие от традиционных языков программирования, таблицы решений в простой форме могут представлять связь между множеством независимых условий и действий.

Таблицы принятия решений, как правило, разделяются на четыре квадранта, как показано ниже.

| | |
|----------|-----------------------------|
| Условия | Варианты выполнения условий |
| Действия | Необходимость действий |

В простейшем случае здесь *Условия* — список возможных условий, *Варианты выполнения условий* — комбинация из выполнения и/или невыполнения условий из этого списка. *Действия* — список возможных действий, *Необходимость действий* — указание надо или не надо выполнять соответствующее действие для каждой из комбинаций условий. Например, для ситуации «неожиданно погас свет» таблица принятия решений может быть такой:

| | | | |
|-------------------------------|----|-----|-----|
| 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 |

Диаграммы Насси - Шнейдермана

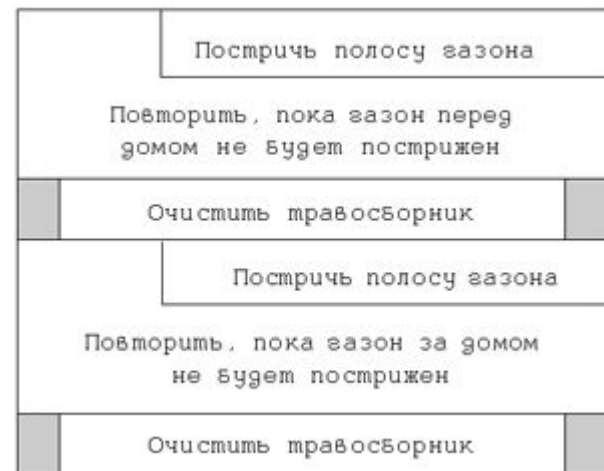
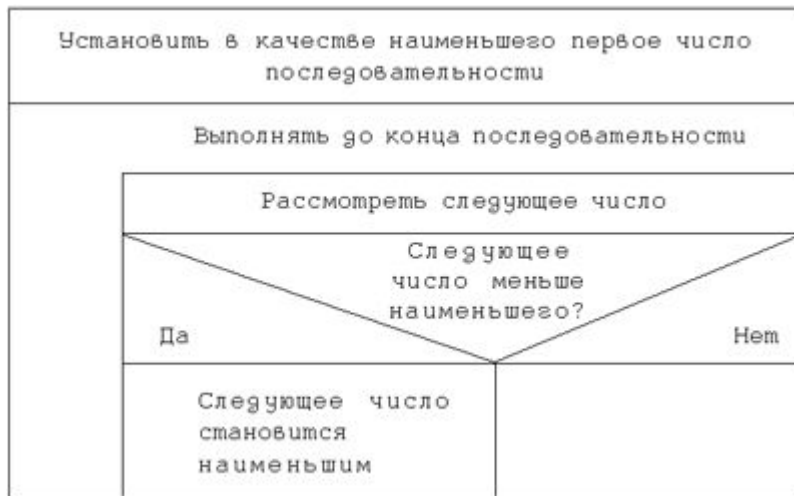


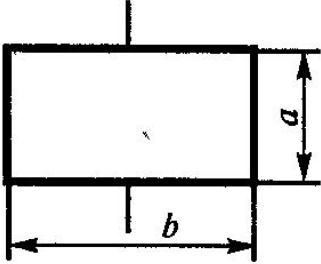
Диаграмма Насси-Шнейдермана Ветвление

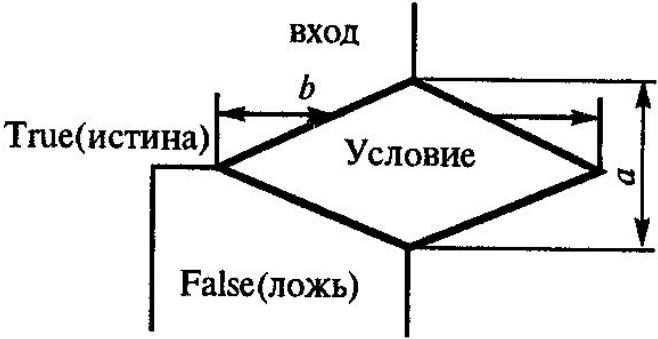


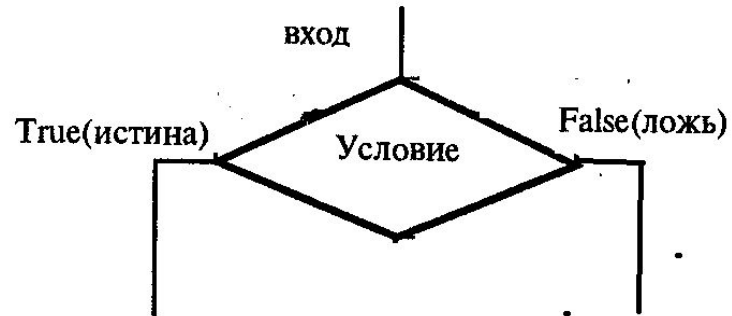
1.2. Составление алгоритма на языке блок-схем

Блок-схема — это графическая интерпретация алгоритма, представляющая набор геометрических фигур, каждая из которых изображает какую-либо операцию или действие. Форма символов и правила составления схем алгоритмов установлены государственными стандартами: ГОСТ 19.701—90 «Схемы алгоритмов, программ, данных и систем».

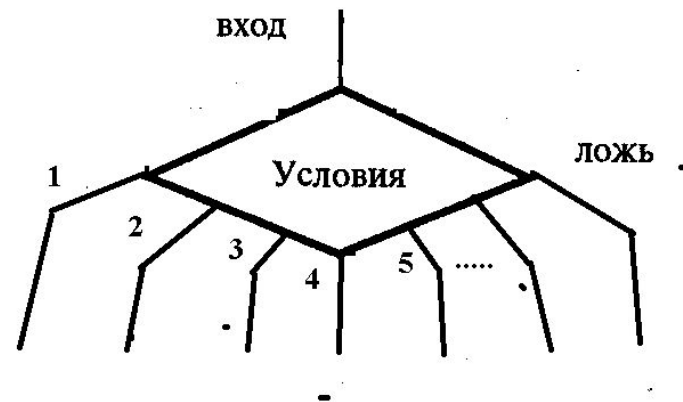
Основные элементы, используемые при построении блок-схем, представлены в табл. 1.

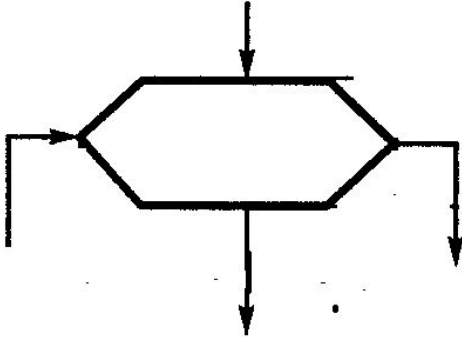
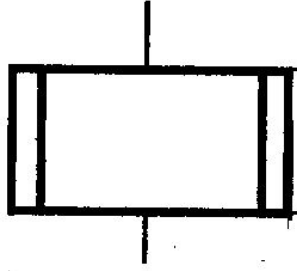
| | | |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| 1 | <p>Процесс — формирование новых значений, выполнение арифметических или логических операций или действий, результаты которых запоминаются в оперативной памяти ЭВМ</p> |  |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|




| | | |
|---|--------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| 2 | <p>Решение — проверка условий: а) выбор одного из двух направлений выполнения алгоритма в зависимости от некоторого условия</p> |  |
|---|--------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|






б) выбор одного из « n » направлений выполнения алгоритма в зависимости от некоторых условий при $n > 2$



| | | |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p>3</p> | <p>Модификация — организация циклических конструкций</p> |  |
| <p>4</p> | <p>Предопределенный процесс — вычисление по подпрограмме, использование ранее созданных и отдельно описанных алгоритмов</p> |  |

| | | |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| 5 | <p>Начало — конец программы или вход и выход в подпрограммах</p> |  |
| 6 | <p>Ввод — вывод данных — связь алгоритма с внешним миром. Вывод может осуществляться на бумагу, на экран монитора, на магнитный диск или ленту</p> |  |
| 7 | <p>Соединитель — разрыв линий потока</p> |  |

| | | |
|----|-----------------------------------------------------------------|--------------------------------------------------------------------------------------|
| 8 | Соединитель — перенос на другую страницу |  |
| 9 | Направление линий потока — стрелки: снизу вверх и справа налево |  |
| 10 | Комментарий — пояснения, содержание подпрограмм |  |

Правила построения алгоритмов на языке блок-схем

1. Блок-схема строится сверху вниз.
2. В любой блок-схеме имеется только один элемент, соответствующий началу алгоритма, и один элемент, соответствующий концу алгоритма.
3. Должен быть хотя бы один путь из начала блок-схемы к любому элементу.
4. Должен быть хотя бы один путь от каждого элемента блок-схемы в конец блок-схемы.

Основные методы современной технологии проектирования алгоритмов

1. Метод структурного проектирования. Любой алгоритм может быть построен из комбинаций трех базовых структур: линейный алгоритм (следование), разветвляющийся алгоритм (развилка) и циклический алгоритм (повтор).

2. Метод нисходящего проектирования. Первоначально выделяются главные функции, затем второстепенные.

3. Метод пошаговой детализации.

4. Метод модульности. Модуль — логически связанный фрагмент программы, выполняющий одну функцию и состоящий из обозримого числа шагов.

Прежде чем приступить к составлению блок-схемы, необходимо:

1. Регламентировать состав входа и выхода, т.е. определить имена входных данных, промежуточных и выходных результатов.

2. Дать наименование основной программе и вспомогательным алгоритмам.

1.3. Базовые управляющие конструкции алгоритмов

1. *Линейные алгоритмы* — последовательность блоков, каждый из которых имеет по одному входу и одному выходу, и выполняется в программе один раз (рис. 4.2, 4.3).

Рассмотрим алгоритм линейной структуры на примере определения площади треугольника по трем известным сторонам a , b и c с использованием теоремы Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \text{ где } p = 0,5 \cdot (a + b + c).$$

2. Алгоритм разветвляющегося вычислительного процесса — алгоритм, в котором в зависимости от значений некоторого признака производится выбор одного из нескольких направлений, называемых ветвями. В основе организации разветвления лежит проверка логического условия, которое может быть истинно или ложно. Частный вид логического условия — это операции типа $=$, \neq , $>$, $<$, \geq , \leq .

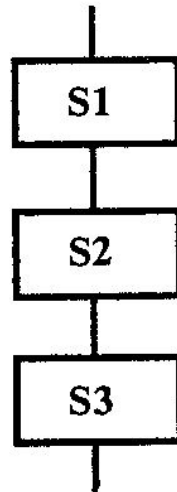


Рис. 2. Алгоритм линейной структуры

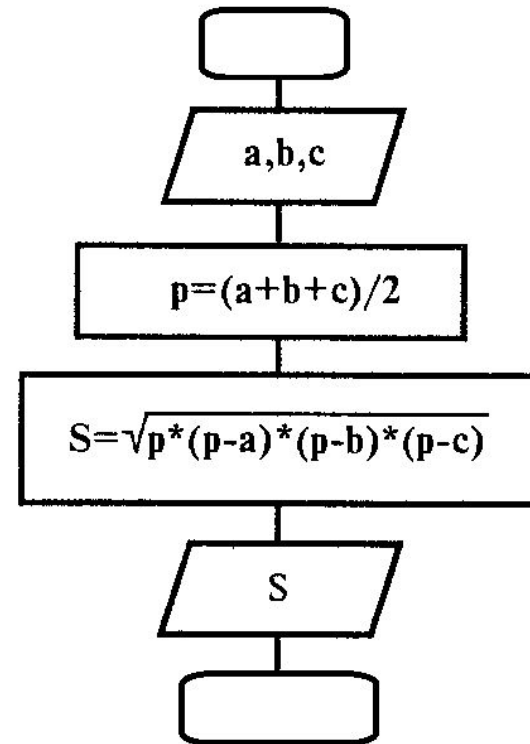


Рис. 3. Алгоритм вычисления площади треугольника по трем сторонам

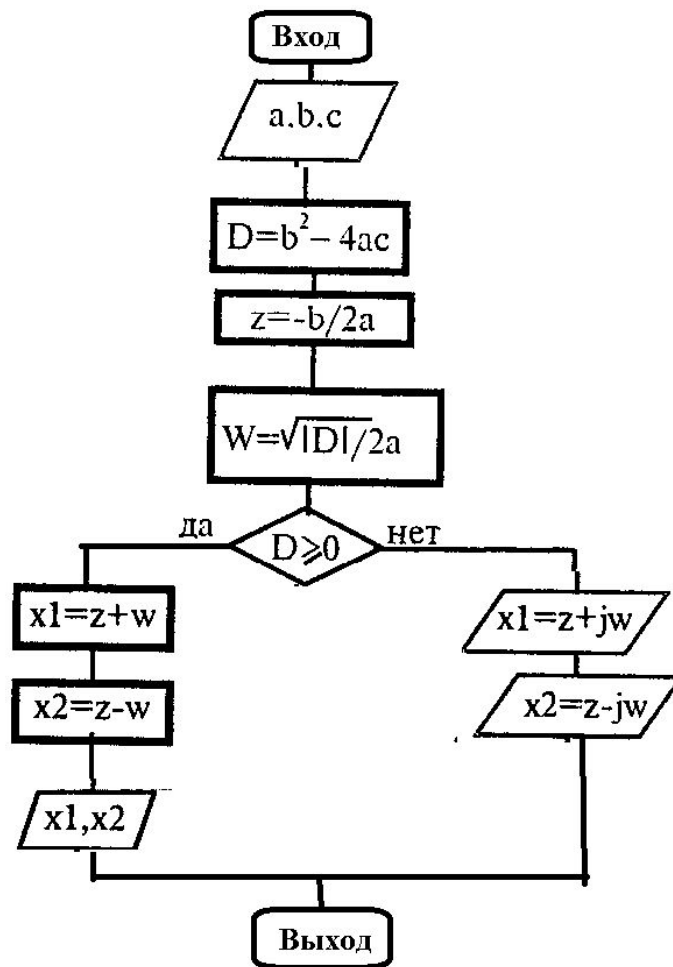
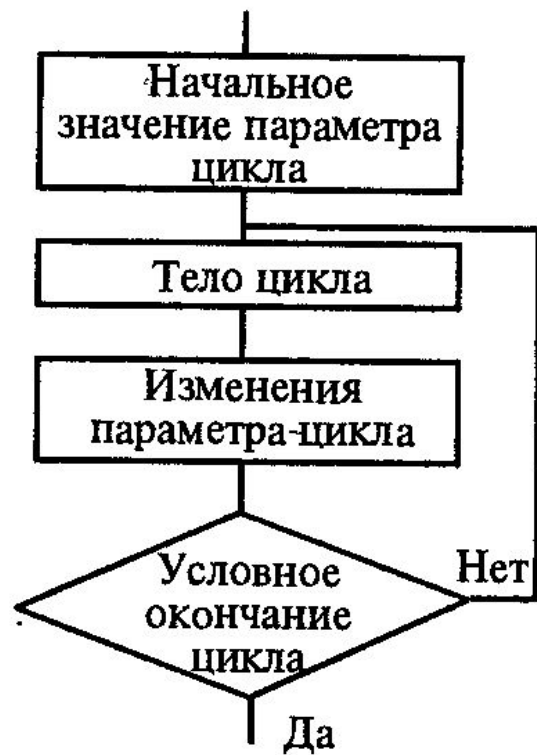


Рис. 6. Блок-схема алгоритма вычисления корней квадратного уравнения



Р и с. 7. Блок-схема арифметического цикла

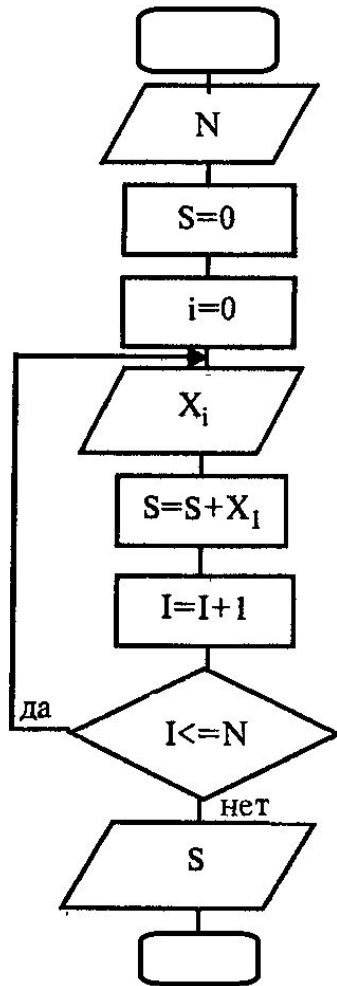


Рис. 8. Блок-схема циклического алгоритма вычисления суммы с использованием блоков «процесс» и «решение»

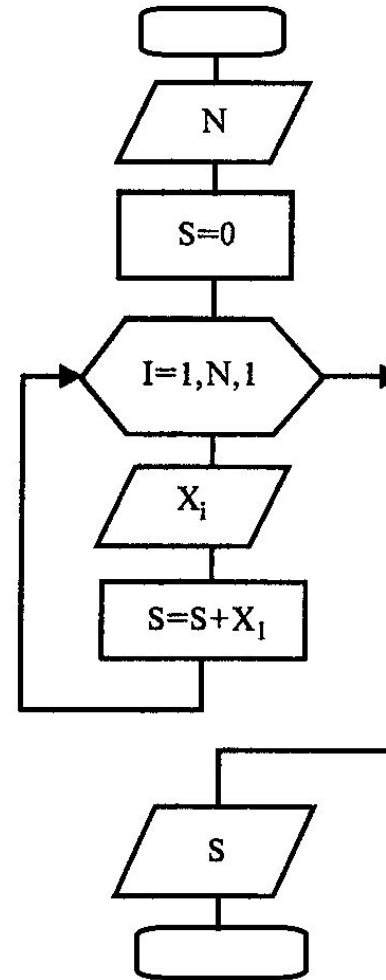


Рис. 9. Блок-схема циклического алгоритма вычисления суммы с использованием блока «модификация»

2. Программирование

2.1. Основные положения

Программирование как процесс создания программы формально состоит из выбора языка программирования и замены элементов блок-схемы алгоритма соответствующими операторами. Правильная программа на алгоритмическом языке представляет собой формальную запись конечной последовательности действий, приводящих к решению поставленной задачи.

Программа, написанная непосредственно в процессорных кодах, представляет собой последовательность из 0 и 1. Команды машинного языка в большинстве случаев состоят из двух частей — из кода операции (указания процессору, *что* сделать), и из операндов (указания, *с чем* нужно сделать операцию). Для ускорения процесса программирования был разработан язык Ассемблер.

При программировании на Ассемблере программа записывается как последовательность строк, начинающихся с имени операции, после которого следуют обозначения операндов. При этом текст программы редактируется как обычный документ. Но такую программу необходимо перевести в машинный язык, в последовательность кодовых слов процессора. Этот этап называется компиляцией и выполняется специальной программой-компилятором, на вход компилятору поступает исходный текст программы, а результатом работы является программа в машинных кодах.

Если компилятор встречает недопустимую комбинацию символов в исходном тексте, он выдает сообщение об ошибке компиляции. От программиста при этом требуется внесение изменений в исходный текст и повторный запуск компилятора.

Поскольку многие программы выполняют одни и те же действия (ввод/вывод данных, вычисление математических функций и т.п.), были организованы библиотеки подпрограмм, где алгоритмы этих действий хранятся уже в скомпилированном виде. При написании программы требуется указать, из какой библиотеки какую подпрограмму нужно вызвать, а связыванием программы и библиотек в единое целое — работоспособную программу — занимается специальная программа-компоновщик, или редактор связей. Компилятор же при этом производит объектный модуль.

В последствии появились языки более высокого уровня, чем Ассемблер. Программы на этих языках состоят уже не из мнемонической записи команд процессора, по одной в каждой строке, а из операторов, каждый из которых также переводится компилятором в машинный код, причем одному оператору может соответствовать несколько машинных инструкций.

Процесс программирования на универсальном языке высокого уровня Паскаль состоит из следующих действий: ввода и редактирования текста программы, трансляции и отладки. Для повышения качества и скорости разработки программ была создана интегрированная система программирования Турбо Паскаль.

Процесс обработки программы на языке Паскаль может быть проиллюстрирован следующей схемой (рис. 13).

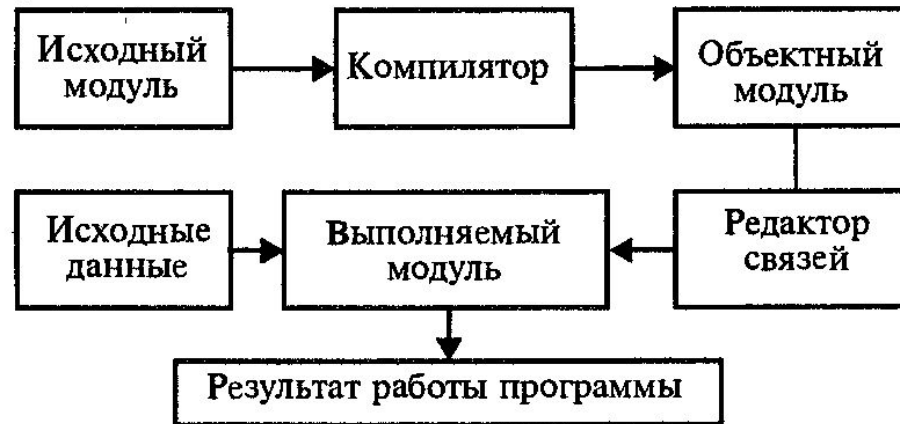


Рис.13. Этапы процесса обработки программы на языке Паскаль

Для выполнения каждого этапа применяются специальные средства интегрированной среды программирования: редактор текстов (editor), компилятор (compiler), компоновщик (linker), отладчик (debugger).

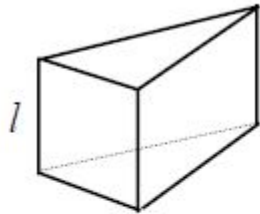
1 ОСНОВНЫЕ ЭТАПЫ РЕШЕНИЯ ЗАДАЧИ НА ЭВМ И ТИПЫ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Обычно выделяются следующие этапы решения задачи на ЭВМ:

- 1) постановка задачи;
- 2) для математических задач - выбор математического метода решения задачи и окончательная математическая постановка задачи (построение математической модели);
- 3) разработка алгоритма решения и наглядное изображение алгоритма в одной из принятых форм;
- 4) написание программы (запись алгоритма на языке программирования);
- 5) ввод текста программы в ЭВМ;
- 6) получение рабочей программы;
- 7) тестирование и отладка программы;
- 8) решение задачи на ЭВМ;
- 9) анализ результатов выполнения программы и оформление их в виде таблиц, графиков и т.д.;
- 10) оформление отчета о проделанной работе.

Рассмотрим выполнение всех этих этапов на примере. Пусть требуется решить на ЭВМ следующую задачу: найти объем прямой треугольной призмы.

1.1 Математическая постановка задачи



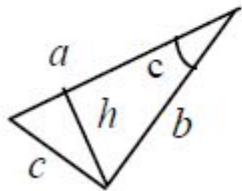
Объем прямой треугольной призмы вычисляется по формуле:

$$V=S*l, \quad (1)$$

где S - площадь основания призмы;

l - высота призмы.

1.1. 2. Выбор математического метода решения задачи



Основанием рассматриваемой призмы является треугольник. Для вычисления площади треугольника существует несколько формул:

$$S=(a*h)/2=(a*b*\sin C)/2= \sqrt{p*(p-a)*(p-b)*(p-c)}.$$

Будем использовать последнюю формулу:

$$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}, \quad (2)$$

где a, b, c - стороны треугольника,

$p = (a+b+c)/2$ - полупериметр треугольника.

Окончательная формулировка задачи: разработать программу для вычисления величины V по формулам (1), (2), если заданы значения a, b, c, l .

Алгоритм решения задачи

Алгоритм решения задачи - это система правил (в виде последовательности арифметических и логических правил), однозначно определяющих процесс преобразования исходных данных в искомые результаты, то есть за конечное число шагов приводящих к решению поставленной задачи.

При составлении алгоритма следует использовать метод пошаговой разработки.

Суть метода: алгоритм разрабатывается "сверху вниз", начиная со списка входных и выходных данных; на каждом шаге принимается небольшое число решений, приводящих к постепенной детализации алгоритма.

Опишем данные, используемые в нашей задаче.

Исходные данные: a, b, c - стороны треугольника, l - высота призмы.

Результаты: V - объем призмы.

Промежуточные данные: p - полупериметр треугольника,
 S - площадь треугольника.

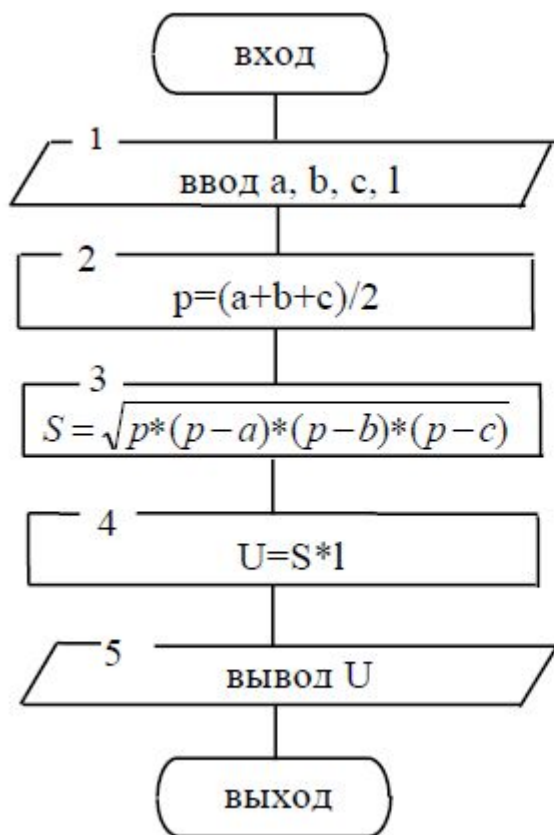
Словесное описание алгоритма обычно громоздко и неудобно для последующего программирования.

Для наглядного графического изображения алгоритма часто используется структурная схема (СС).

СС состоит из отдельных блоков (геометрических фигур), соединенных между собой.

Форма геометрической фигуры характеризует функции, выполняемые соответствующим блоком.

Внутри фигуры словесно или с помощью формул эти функции конкретизируются.



1. Ввод в память ЭВМ значений переменных a,b,c,l.
2. Вычисление значения переменной p по формуле: $p=(a+b+c)/2$.
3. Вычисление значения переменной S по формуле: $S= p*(p-a)*(p-b)*(p-c)$.
4. Вычисление значения переменной V по формуле: $V=S*1$.
5. Вывод значения переменной V

Программирование - это запись разработанного алгоритма на языке программирования (ЯП).

В ЯП для указания выполняемых действий служит оператор. Программа на ЯП – это последовательность операторов, оформленная по специальным правилам.

В дальнейшем в качестве ЯП используется Паскаль.

Писать программу следует в соответствии со СС.

Каждому блоку СС соответствует обычно один или несколько операторов ЯП.

Ниже дана программа для рассматриваемой задачи.

```
program vtp(i,o);
var
    a,b,c,l,p,s,v:real;
begin
    writeln('Вычисление объема прямой треуг. призмы');
    write('Введите a,b,c - стороны треугольника: ');
    read(a,b,c);
    write('Введите l - высоту призмы: ');
    read(l);
    p:=(a+b+c)/2;
    s:=sqrt(p*(p-a)*(p-b)*(p-c));
    v:=s*l;
    writeln('Объем призмы v=',v)
end.
```

Ввод текста программы в ЭВМ

Этот этап связан с навыками работы с вычислительной техникой. При работе на персональном компьютере данные (текст программы, исходные данные) вводятся с помощью клавиатуры.

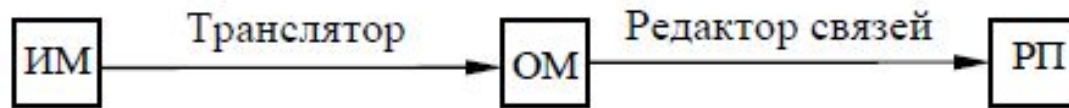
Вводимые данные, а также результаты выполнения программы, отображаются на экране дисплея.

Получение рабочей программы

Текст программы на ЯП называется исходным модулем (ИМ). ИМ не может непосредственно выполняться на ЭВМ.

С помощью транслятора (компилятора) и редактора связей должна быть создана рабочая программа (РП), соответствующая данному ИМ.

РП - это программа, готовая к выполнению на ЭВМ. Ниже схематично изображен порядок создания РП.



Трансляция (компиляция) ИМ включает в себя два действия: анализ – определение правильности записи программы в соответствии с правилами ЯП и синтез – генерирование соответствующей программы на машинном языке, которая называется объектным модулем (ОМ).

В процессе анализа компилятор находит синтаксические ошибки в программе и выдает сообщения о них. После устранения ошибок пользователь должен вновь откомпилировать программу. Если ошибок нет, то компилятор вырабатывает ОМ, эквивалентный ИМ.

Программист может использовать в своей программе другие программы. Редактор связей соединяет все ОМ и создает готовую для выполнения на ЭВМ РП.

Транслятор и редактор связей - это системные программы, входящие в систему программирования и предоставляемые пользователю в готовом виде; в случае необходимости пользователь вызывает их для выполнения.

Тестирование и отладка программы

Отсутствие синтаксических ошибок еще не означает, что программа написана правильно. Существуют ошибки, которые обнаруживаются только на этапе выполнения программы.

Например, в программе `vtp` (см. слайд 36)

вместо оператора `"v:=s*I`

мог быть

записан оператор `"v:=s+I;"`.

С точки зрения синтаксиса эта программа была бы верна, но на самом деле она будет выдавать неверный результат.

Для обнаружения в программе как можно большего количества ошибок проводится тестирование программы.

Тестирование заключается в составлении набора тестов (как правильных, так и неправильных) и выполнении программы на этих тестах.

Простейший тест - это набор исходных данных, для которых известен результат.

Тест, приводящий к неправильному выполнению программы, свидетельствует о наличии в программе ошибки.

Далее необходимо локализовать местонахождение ошибки и исправить ее. Этот процесс называется отладкой.

При исправлении одних ошибок в программу могут вноситься другие ошибки, поэтому исправленную программу необходимо вновь подвергнуть тестированию и т.д.

В результате большинство (но, возможно, не все) ошибки будут удалены из программы, после чего можно проводить решение задачи на ЭВМ.

Решение задачи на ЭВМ и анализ результатов

Этот этап заключается в следующем: готовятся исходные данные для программы;

программа запускается, производит необходимые действия и выдает полученные результаты.

Пользователь анализирует эти результаты и оформляет их в надлежащем виде.

1. 9. Оформление отчета о проделанной работе

Отчет о проделанной работе (в частности, отчет по лабораторной работе) должен содержать следующие разделы:

- 1) заголовок работы;
- 2) формулировка задачи;
- 3) математическая постановка задачи (при необходимости);
- 4) обозначение входных, выходных и промежуточных данных;
- 5) алгоритм решения задачи, представленный в виде СС или словесного описания;
- 6) текст программы на языке ЯП;
- 7) набор тестов для проверки программы и результаты отладки программы;
- 8) результаты выполнения программы на заданных исходных данных (для реальных прикладных программ).

Обозначение входных, выходных и промежуточных данных можно не оформлять как отдельный раздел отчета, а оформить обозначения в виде комментариев в разделе описаний в тексте программы.

1.2 ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА ПАСКАЛЬ

Автором языка Паскаль (Pascal) является профессор Никлас Вирт, специалист по вычислительной технике, директор института информатики в Цюрихе. Вирт опубликовал описание своего языка в 1997 г. Язык назван "Паскаль" в честь великого французского ученого 17-го века, который первый в мире изобрел автоматическое устройство для сложения чисел.

Паскаль - один из самых распространенных языков программирования, имеет много достоинств, среди которых выделим:

- простота,
- мощные средства обработки данных,
- ориентация на принцип современной технологии программирования.

В настоящий момент язык Паскаль реализован почти на всех мини- и микро-ЭВМ, что и определило его широкое распространение. Ниже дано описание языка применительно к версии Turbo-Pascal 7.0, однако во многих случаях изложение ведется в стиле классического языка Паскаль, что обеспечивает совместимость с разными версиями языка.

1.2.2 Алфавит

Любые конструкции языка строятся из символов алфавита. В Паскале выделено 4 типа символов (знаков):

1) буквы (26 букв латинского алфавита): A,B,C,D,...Z; для отечественных ЭВМ допускается в комментариях использовать русские буквы;

2) цифры (9 арабских цифр): 0,1,2,3,...9;

3) специальные символы:

- знаки арифметических операций:

"+", "-", "*", "/",

- знаки операций отношения:

">", "<", "<=", ">=", "=", "<>",

- операция присваивания:

":=",

- разделители и ограничители:

""", ":", ";", ".", "(", ")", "{", "}"

4) ключевые (зарезервированные, служебные) слова; использовать эти слова в каком-либо другом значении, кроме того, которое зафиксировано в Паскале, нельзя; примеры ключевых слов:

and, array, begin, end, program.

1.2.3 Идентификатор

Идентификатор (имя) используется для обозначения объектов программы, а именно программы, переменной, константы, типа и т.д.

Идентификатор - это произвольная последовательность букв, цифр и знаков подчеркивания, начинающаяся с буквы или знака подчеркивания. Длина идентификатора произвольная, однако компилятор различает имена по первым 63 символам. Рекомендуется ограничивать длину идентификатора 8 символами, что обеспечивает совместимость с любыми версиями Паскаля.

Синтаксическая диаграмма идентификатора (знак подчеркивания рассматривается как буква):



Замечание 1. Различают два вида идентификаторов: стандартные и даваемые пользователем. Стандартные имена заложены в языке для обозначения стандартных объектов (например, стандартных функций SIN, COS и т.д.) Когда пользователь называет в своей программе какой-то объект, то он не должен использовать ключевые слова и стандартные идентификаторы.

Замечание 2. Строчные и прописные буквы в идентификаторах и ключевых словах не различаются, то есть, например, имена NIME и nime - одно и то же.

1.2.4 Структура программы

1.2.4.1 Программа на языке Паскаль состоит из заголовка, раздела описаний (объявлений) и раздела операторов.

```
PROGRAM <имя программы> ( INPUT,OUTPUT );  
    <раздел объявлений>  
BEGIN  
    <раздел операторов>  
END.
```

Заголовок содержит служебное слово "PROGRAM", имя программы, задаваемое программистом, а в круглых скобках - список параметров. Выше указан список "INPUT, OUTPUT", который обычно записывают короче: (i,o). INPUT, OUTPUT - это имена стандартных процедур ввода и вывода на терминал, соответственно. INPUT указывается, если в программе есть ввод данных; OUTPUT указывается всегда. Иногда список параметров длиннее:

```
PROGRAM N ( INPUT, OUTPUT, X,Y, .... );
```

Здесь X,Y - внешние файлы, используемые в программе.

Заканчивается заголовок знаком ";".

В Turbo Pascal заголовок, а, значит, и список параметров, указывать не обязательно.

1.2.4.2 Раздел объявлений служит для описания данных, встречающихся в программе. Этот раздел состоит из пяти подразделов:

LABEL - объявление меток;

CONST - объявление констант;

TYPE - объявление типов;

VAR - объявление переменных;

PROCEDURE, FUNCTION - объявление процедур и функций.

В классической версии Паскаля эти подразделы должны располагаться в строго определенном порядке, но при этом некоторые из них могут отсутствовать. Каждый подраздел начинается специальным ключевым словом, которое записывается один раз в начале каждого подраздела. После каждого объявления ставится ";".

В Turbo Pascal порядок размещения подразделов произвольный, может быть несколько одинаковых подразделов.

1.2.4.3 Раздел операторов заключается в операторные скобки вида BEGIN (начать) и END (кончить). После END обязателен знак "."(точка). В этом разделе записывается выполняемая часть программы, состоящая из операторов (команд, действий, инструкций). Обычно операторы отделяются друг от друга знаком ";".

1.2.4.5 Пример полностью законченной программы на Паскале.

Задача. Вычисление объема цилиндра по формуле $V = \pi \cdot R^2 \cdot H$.

Дано: R - радиус,

Н - высота.

Результат: V - объём.

Программа предназначена для работы в режиме диалога с ЭВМ. В этом режиме пользователь вводит исходные данные, набирая их на клавиатуре дисплея, и получает от ЭВМ результаты работы программы на экране дисплея.

```
(* *)
(* Вычисление объёма цилиндра *)
(* *)
PROGRAM PRIMER (INPUT,OUTPUT);
  CONST PI=3.14; (* число  $\pi$  *)
  VAR R:REAL; (* радиус цилиндра *)
      H:REAL; (* высота цилиндра *)
      V:REAL; (* объем цилиндра *)
BEGIN (* начало основной программы *)
  READ(R,H); (* Ввод двух чисел с клавиатуры и запись их в переменные R и H *)
  V:=PI*R*R*H; (* Вычисление объема цилиндра *)
  WRITELN(V); (* Вывод на экран значения переменной V - вычисленного объема *)
END. (* конец программы *)
(* *)
```

1.2.5 Элементы данных: константы и переменные

Основные вычисления в программе осуществляются над данными, то есть над константами и переменными.

1.2.6 Типы данных

Каждое данное имеет тип, который определяет: внутримашинное представление значение данного; количество байтов, отводимых для хранения этого значения, и набор операций, которые выполняются со значением данного.

Рассмотрим 4 стандартных типа Паскаля:

Integer, Real, Boolean, Char.

1.2.6.1 Целый тип *INTEGER*

Данные целого типа принимают только целочисленные значения в диапазоне [-32768;+32768]. Для положительных величин знак "+" можно опускать.

С данными этого типа могут выполняться следующие операции:

"+" - сложение,

"-" - вычитание,

"*" - умножение,

" div " - деление нацело,

" mod " - остаток от деления нацело.

В памяти ЭВМ данное целого типа занимает 2 байта.

Пример. Предположим, в программе есть две переменные целого типа A и B и в текущий момент они имеют значения: A=7, B=3. Тогда (знаком "=>" обозначен результат выполнения операции):

A+B => 10

A-B => 4

A*B => 21

A div B => 2

A mod B => 1

В отличие от классического языка Паскаль, где определен единственный целый тип Integer, в Turbo Pascal имеется 5 целых типов: Shortint, Integer, Longint, Byte, Word. Они различаются диапазоном, форматом (знаковый, беззнаковый) и размером в байтах.

1.2.6.2 Вещественный (действительный, реальный) тип REAL

Данные вещественного типа могут принимать дробные значения в диапазоне от $2.9 \cdot 10^{-38}$ до $1.7 \cdot 10^{38}$.

При записи вещественных чисел могут использоваться две формы:

- 1) обычная, когда дробная часть отделяется от целой десятичной точкой: -3.2; 10.46546.
- 2) экспоненциальная, когда число записывается с десятичным порядком: -2.51E+14 или -2.51E14 соответствует $-2.51 \cdot 10^{14}$.

Число, записанное в экспоненциальной форме, состоит из двух частей, разделенных латинской буквой E или e. Слева от E (e) указывается мантисса - вещественное число в обычной форме, а справа указывается порядок - целое число (возможно, со знаком), содержащее не более двух цифр. Чтобы получить значение числа, следует мантиссу умножить на 10 в степени, равной порядку. Экспоненциальная форма удобна при записи очень маленьких или очень больших чисел.

С данными вещественного типа могут выполняться операции:

- "+" - сложение,
- "-" - вычитание,
- "*" - умножение,
- "/" - деление,

причем хотя бы один операнд должен быть вещественного типа, другой может быть как вещественный, так и целочисленный.

В памяти ЭВМ данное вещественного типа занимает 4 байта.

В отличие от классического языка Паскаль, где определен единственный вещественный тип Real, в Turbo Pascal имеется 5 вещественных типов: Real, Single, Double, Extended, Comp. Они различаются диапазоном, числом значащих цифр и размером в байтах.

1.2.6.3 Булевский (логический) тип *BOOLEAN*

В языке Паскаль имеются две логические (булевские) константы: `TRUE` и `FALSE`, истина и ложь, соответственно.

Логическая переменная может принимать одно из этих двух значений.

Для данных булевского типа разрешены операции (подробно рассматриваются в п. 2.3):

`AND` - логическое "и",

`OR` - логическое "или",

`NOT` - логическое "нет".

Логические данные имеют важное значение в информатике. Они используются при построении условий во многих операторах. Часто в программах переменную объявляют как булевскую, если она должна указывать на то, произошло некоторое событие или нет (переменная - "флажок").

В памяти ЭВМ данное булевского типа занимает 1 байт.

В отличие от классического языка Паскаль, где определен единственный вещественный тип `Real`, в `Turbo Pascal` добавлено еще 3 логических типа: `ByteBool`, `WordBool`, `LongBool`. Они различаются размером в байтах и фактической величиной, соответствующей значению `TRUE`.

1.2.6.4 Символьный (знаковый, литеральный) тип CHAR

Символьная константа - это символ, заключенный в одиночные апострофы. Например: 'A', 'a', '1', '+'. Можно использовать любые символы, которые имеются на данной вычислительной машине, то есть обеспечивается полный набор ASCII-символов.

Символьная переменная принимает значение символьной константы.

В памяти ЭВМ данное символьного типа занимает 1 байт.

В Паскаль-программе можно использовать константы-строки. Это последовательность символов, заключенная в одиночные апострофы. Например:

'Hello!'.

1.2.7 Стандартные подпрограммы. Стандартные арифметические функции

Рассмотрим наиболее распространенные стандартные арифметические функции. Они имеют один входной параметр, или аргумент. Обозначим его как x . Аргумент x - числовой, типа REAL или INTEGER. Ниже приведен список этих функций в виде таблицы.

| Название функции | Описание функции | Тип результата |
|------------------|------------------|-----------------------|
| abs(x) | $ x $ | тот же, что и тип x |
| sqr(x) | x^2 | тот же, что и тип x |
| sqrt(x) | \sqrt{x} | real |
| sin(x) | sinx | real |
| cos(x) | cosx | real |
| arctan(x) | arctgx | real |
| exp(x) | e^x | real |
| ln(x) | lnx | real |
| pi | π | real |

В тригонометрических функциях \sin , \cos , arctg аргумент задается в радианах, поэтому для перевода градусов в радианы необходимо использовать формулу:

$$x(\text{рад}) = x(\text{град}) \cdot \pi / 180.$$

В Паскале отсутствует стандартная подпрограмма для вычисления аргумента x в произвольной степени n , то есть x^n . В этом случае можно воспользоваться соотношением:

$$x^n = e^{n \cdot \ln(x)},$$

которое справедливо для $x > 0$.

Для получения произвольных логарифмов использовать формулу:

$$\log_b a = \frac{\log_c a}{\log_c b}, \quad \text{или} \quad \log_b a = \frac{\ln a}{\ln b}$$

Для вычисления различных обратных тригонометрических функций через арктангенс можно использовать формулы:

Для арксинуса:

$$\arcsin x = \text{arctg} \frac{x}{\sqrt{1-x^2}}$$

Для арккосинуса:

$$\arccos x = \frac{\pi}{2} - \arcsin x = \frac{\pi}{2} - \text{arctg} \frac{x}{\sqrt{1-x^2}}$$

Для арккотангенса:

$$\operatorname{arccotg} x = \frac{\pi}{2} - \operatorname{arctg} x$$

Любой корень можно получить используя формулу:

$$\sqrt[n]{x} = x^{\frac{1}{n}} = e^{\ln x^{\frac{1}{n}}} = e^{\frac{1}{n} \ln x}$$

1.2.8 Объявление констант и переменных

1.2.8.1 Объявление констант

Константа в программе может быть задана явно своим значением или обозначена именем. В последнем случае константа должна быть описана в разделе объявления констант.

Синтаксис раздела

```
CONST <имя константы>=<значение>;{<имя константы>=<значение>;}
```

Пример:

```
const pi=3.14; (* число  $\pi$  *)  
    nim=5; (* номер варианта *)
```

Обычно константу обозначают именем, если она имеет громоздкую запись или это изменяемая константа (вариант программы). В этом случае программа становится короче и наглядней.

1.2.8.2 Объявление переменных

Пример:

```
var h:real;    (* h - вещественная переменная *)  
    r:real;    (* r - вещественная переменная *)  
    b:integer; (* b - целочисленная переменная *)  
    c:integer; (* c - целочисленная переменная *)
```

Эти же переменные можно описать короче, перечислив однотипные переменные через запятую:

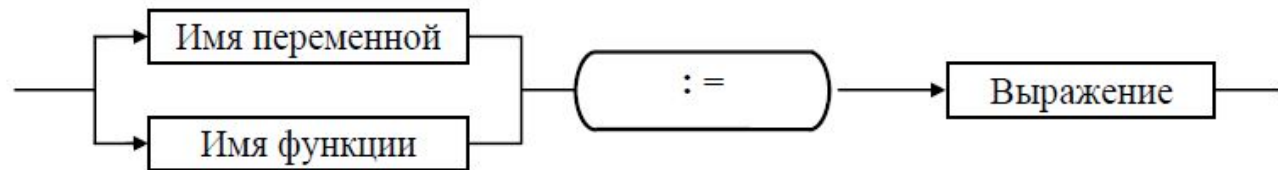
```
var h,r:real;  (* h,r - вещественные переменные *)  
    b,c:integer; (* b,c - целочисленные переменные *)
```

1.3.1 Оператор присваивания

1.3.1.1 Синтаксис оператора присваивания

Оператор присваивания - основной оператор любого языка программирования. Он означает, что вновь вычисленное значение присваивается некоторой переменной. Имя этой переменной записывается слева от операции присваивания ":=". Справа от этого знака записывается выражение, с помощью которого вычисляется значение переменной.

Синтаксис оператора присваивания в виде СД:



Знак присваивания ":= " читается "становится равным".

Примеры.

a:=1; (* Переменной a присваивается значение, равное 1 *)

2:a:=1; (* Переменной a присваивается значение, равное 1. Этот оператор присваивания помечен меткой 2 *)

b:=b+1; (* Новое значение переменной b равно ее старому значению плюс 1 *)

Операцию присваивания ":= " не следует путать с операцией сравнения "=". Операция ":= " присваивает переменной новое значение, операция "=" сравнивает две величины.

1.3.1.2 Выражение

Выражение состоит из операций и операндов. Операндом может быть константа, переменная, указатель функции.

В таблице ниже приведены примеры записи выражений на языке Паскаль.

| Обычная запись | Запись на Паскале |
|----------------|-------------------|
| 4,5 | 4.5 |
| ab+c | a*b+c |
| a:b | a/b |
| (a+b)sinx | (a+b)*sin(x) |
| x^5 | exp(5*ln(x)) |

Правила записи выражений:

- 1) нельзя записывать подряд два знака операции, необходимо разделять их скобками: $a+(-b)$, $a/(-z)$;
- 2) нельзя опускать знак умножения, необходимо обозначать его знаком "*": $a*b$;
- 3) операции "+", "*", "-", "/" необязательно окружать пробелами, операции "div" и "mod" обязательно должны быть окружены пробелами: $a \text{ div } b$, $c \text{ mod } 2$.

Порядок вычисления выражения:

- 1) если выражение не содержит скобок, операции выполняются в следующем порядке:
 - а) +, - (унарные), not
 - б) *, /, div, mod, and
 - в) +, -, or
 - г) =, <, >, <=, >=, in

↓
убывание
приоритета
- 2) операции одинакового приоритета выполняются слева направо;
- 3) операции в круглых скобках выполняются в первую очередь, и с помощью круглых скобок можно задать любой требуемый порядок вычислений.

Пример 1. Выражение $15 \operatorname{div} 4 * 4$ эквивалентно выражению $(15 \operatorname{div} 4) * 4$, так как операции "div" и "*" одинакового приоритета и выполняются поэтому слева направо.

Пример 2. Выражение $4 + 15 \operatorname{div} 4$ эквивалентно выражению $4 + (15 \operatorname{div} 4)$, так как у операции "div" приоритет выше, чем у бинарной операции "+".

1.3.1.3 Тип выражения

Любое выражение имеет тип, который определяется как тип значения, получающегося в результате вычисления выражения.

Различают логические выражения (типа `boolean`) и арифметические выражения (типа `real` или `integer`).

Результатом логического выражения является значение `true` или `false`, логическое выражение можно присвоить логической переменной.

Результатом арифметического выражения является числовая величина. Тип арифметического выражения (`integer` или `real`) всегда можно определить до начала вычисления, исходя из следующего:

- 1) тип переменных и констант задается при их объявлении;
- 2) для операций "+", "-", "*" - если хотя бы один операнд типа `real`, то и результат будет типа `real`, если оба операнда типа `integer`, то и результат будет типа `integer`;
- 3) для операции "/" результат всегда будет типа `real`;
- 4) для операций "div", "mod" - операнды всегда должны быть типа `integer`, результат также будет типа `integer`.

В операторе присваивания переменная и выражение должны быть одного типа. Исключение: допустим случай, когда переменной типа `real` присваивается значение выражения типа `integer`. Все остальные случаи несоответствия по типу для оператора присваивания в Паскале недопустимы.

Пример. Предположим, даны следующие описания переменных:

var i,j,k:integer; (* i,j,k - целые переменные *)

a,b,c:real; (* a,b,c - вещественные переменные *)

bool:boolean; (* bool - логическая переменная *)

c1,c2:char; (* c1,c2 - знаковые переменные *)

Тогда можно записать следующие операторы присваивания:

i:=i div k+j; (* Целой переменной i присваивается результат целого выражения *)

a:=b/c; (* Вещественной переменной a присваивается результат вещественного выражения *)

a:=k*j div 2; (* Вещественной переменной a присваивается результат целого выражения *)

bool:=a=b; (* Логической переменной bool присваивается результат логического выражения "a=b" *)

c1:='4'; (* Знаковой переменной c1 присваивается значение знака '4' *)

c2:=c1; (* Знаковой переменной c2 присваивается значение другой знаковой переменной c1 *)

Как уже отмечалось выше, присвоить целой переменной вещественный результат нельзя. В подобном случае для округления и получения целого результата используются стандартные функции `round` и `trunc`, которые преобразуют тип `real` в `integer`. Пусть `x` - переменная типа `real`, тогда `round(x)` дает ближайшее целое к `x` (округляет до ближайшего целого), `trunc(x)` дает целую часть `x` (обрезает до ближайшего целого).

Пример.

`round(5.9)` дает целый результат 6.

`trunc(5.9)` дает целый результат 5.

1.3.2 Простейшие операторы ввода-вывода

1.3.2.1 Операторы ввода: read и readln

Операторы ввода позволяют ввести в программу исходную информацию с клавиатуры. Общий формат операторов:

```
read(a1,a2,a3,...,an);  
readln(a1,a2,a3,...,an);
```

где a1,a2,a3,...,an - список ввода, то есть список имен переменных, значения которых необходимо ввести;

read и readln - имена стандартных ПП (процедур) ввода.

В общем случае оператор ввода выполняется следующим образом: ЭВМ приостанавливает свою работу и ждет, пока с клавиатуры не будут введены соответствующие для списка ввода значения и нажата клавиша "Ввод" (<ВК>).

Пример. Пусть для следующих описаний переменных:

```
var x,y:real;
```

Пример. Пусть для следующих описаний переменных:

```
var x,y:real;  
    n:integer;
```

имеем следующий оператор ввода:

```
read(x,y,n).
```

Соответствующий ввод с клавиатуры можно провести двумя способами:

1) разделяя значения пробелом, например:

```
2.5 -1.05 10<BK> ;
```

2) разделяя значения клавишей <BK> и размещая тем самым значения на отдельных строках, например:

```
2.5<BK>  
-1.05<BK>  
10<BK>
```

И в том, и в другом случае переменной x присвоится значение 2.5, переменной y - значение -1.05, переменной n - значение 10.

readln без параметров. В этом случае:

1) если входная строка не пустая, то она очистится,

2) если входная строка пустая, система будет ждать ввода с клавиатуры, который должен завершиться клавишей <ВК>; затем введенная информация проигнорируется, а входная строка очистится.

Пример. В программе необходимо ввести значения шести целых переменных a,b,c,d,e,f. Рассмотрим 3 варианта реализации ввода.

Вариант 1.

Предположим, при организации в программе ввода в виде

```
readln(a,b,c);
```

```
read(d,e,f);
```

или

```
read(a,b,c);
```

```
read(d,e,f);
```

ввод с клавиатуры произведен следующим образом

```
10 20 30<ВК>
```

```
40 50 60<ВК>
```

Тогда и в первом, и во втором случае переменным присвоятся следующие значения: a=10, b=20, c=30, d=40, e=50, f=60.

Вариант 2.

Предположим, при организации в программе ввода в виде

```
readln(a,b,c);  
read(d,e,f);
```

ввод с клавиатуры произведен следующим образом

```
10 20 30 40 50 60<ВК>
```

Тогда, в соответствии с первым оператором ввода `readln`, переменным `a`, `b`, `c` присвоятся значения: `a=10`, `b=20`, `c=30`. Оставшиеся значения в строке ввода будут проигнорированы, и при выполнении второго оператора ввода `read` система будет ждать ввода с клавиатуры новых значений для переменных `d`, `e`, `f`.

Вариант 3.

Предположим, при организации в программе ввода в виде

```
read(a,b,c);
```

```
read(d,e,f);
```

ввод с клавиатуры произведен следующим образом

```
10 20 30 40 50 60<ВК>
```

Тогда, в соответствии с первым оператором ввода `read`, переменным `a`, `b`, `c` присвоятся значения: `a=10`, `b=20`, `c=30`. При выполнении второго оператора ввода `read` система не будет ждать ввода с клавиатуры значений для переменных `d`, `e`, `f`, а возьмет для этих переменных оставшиеся значения из строки ввода: `d=40`, `e=50`, `f=60`.

Правила ввода данных с клавиатуры:

- 1) вводимые величины отделяются друг от друга пробелами или размещаются с помощью клавиши <ВК> на отдельных строках;
- 2) внутри вводимого числа пробелы недопустимы;
- 3) для указания окончания ввода нажимается клавиша <ВК>;
- 4) вводимые величины должны соответствовать переменным из списка ввода по типу, для переменных типа `real` допустимо вводить значения типа `integer`;
- 5) значения логических переменных (`true` и `false`) ввести нельзя;
- 6) при вводе знаковых значений для переменных типа `char` апострофы не нужны;
- 7) нажатие клавиши <ВК> означает ввод двух неотображаемых знаков: "возврат каретки" и "перевод строки";
- 8) следует внимательно относиться ко вводу знаков, особенно ко вводу знаков после ввода числовых значений (см. примеры ниже),
- 9) во избежании непонимания со стороны пользователя и вследствие этого неправильного ввода данных, программа всегда должна перед вводом выдавать сообщение о том, что, в каком виде и как следует ввести.

Пример 1. Для трех знаковых переменных a, b, c необходимо ввести следующие значения: a='1',b='2',c='3'. В программе ввод запрограммирован как "readln(a,b,c);" или "read(a,b,c);". В этом случае правильный ввод с клавиатуры - это ввод знаков подряд без пробелов и <ВК> между ними:

123<ВК>

Неправильным будет ввод знаков через пробел:

1 2 3<ВК>

В этом случае переменным присвоятся следующие значения: a='1', b=' ' (знак "пробел", которым разделены знаки '1' и '2'), c='2'.

Также неправильным будет ввод знаков через клавишу <ВК>:

1<ВК>

2<ВК>

3<ВК>

В этом случае переменной a присвоится знак '1', а переменным b и c -неотображаемые знаки "возврат каретки" и "перевод строки", соответствующие клавише <ВК>.

Пример 2. Часто ошибки возникают при попытке ввести знак сразу после числа. Предположим, для вещественной переменной `r` и двух знаковых переменных `c1`, `c2` необходимо ввести следующие значения: `r=5.2`, `c1='f'`, `c2='1'`. Если в программе ввод значений этих трех переменных будет запрограммирован одним оператором `"readln(r,c1,c2)"` или `"read(r,c1,c2)"`, то ввести корректно соответствующие значения с клавиатуры будет невозможно. Встанет проблема, как при вводе отделить число `5.2` от следующего знака `'f'`: если в качестве разделителя будет пробел или клавиши `<ВК>`, то возникнет ошибка, рассмотренная в предыдущем примере; если знак ввести сразу после числа в виде `"5.2f1"`, то возникнет ошибка выполнения, связанная с неправильным числовым форматом.

Итак, не следует одним оператором вводить и числа, и знаки. Перед оператором ввода знака следует обязательно использовать оператор `readln` для очистки входной строки. Задачу, рассмотренную в примере 2, можно решить, разбив ввод на два последовательных оператора:

```
readln(r);  
readln(c1,c2);
```

1.3.2.2 Операторы вывода: *write* и *writeln*

Операторы вывода позволяют распечатать информацию на экране дисплея. Общий формат операторов:

```
write(a1,a2,a3,...,an);
```

```
writeln(a1,a2,a3,...,an);
```

где $a_1, a_2, a_3, \dots, a_n$ - список вывода, в котором отдельный элемент - это или имя

переменной, значение которой выводится на экран, или строка-константа

в одиночных апострофах, которая "один к одному" отображается на экране;

write и *writeln* - имена стандартных ПП (процедур) вывода.

Оператор *writeln*, кроме того, что распечатает все данные в соответствии со списком вывода, переведет после печати курсор на начало следующей строки.

Возможно использовать ПП *writeln* без параметров. В этом случае курсор на экране перейдет на начало следующей строки.

Пример 1. Предположим, текущее значение двух целочисленных переменных m и n равно 10 и 20, соответственно: $m=10$, $n=20$. Тогда при печати значений этих переменных в виде

```
write(a);write(b);
```

на экране появится результат в одной строке: 1020

При печати в виде

```
writeln(a);write(b);
```

на экране появится результат в двух строках:

```
10
```

```
20
```

Пример 2. Предположим, текущее значение целочисленной переменной m равно 10: $m=10$. Тогда при печати значения этой переменной в виде

```
write('Результат: y=',y);
```

на экране появится результат в одной строке в виде:

```
Результат: y=10
```

1.3.2.3 Формат вывода

При выводе для величин определенного типа данных отводится по умолчанию определенное число позиций:

1) `integer` - под число отводится то число позиций, которое соответствует количеству цифр в числе; для отрицательных величин добавляется еще одна позиция под знак "минус";

2) `real` - число распечатывается в экспоненциальном представлении и занимает обычно 17 позиций (число позиций зависит от конкретной реализации системы): 13 позиций - мантисса (из них 10 позиций - дробная часть мантиссы), 4 позиции - знак "Е" и порядок;

3) `char` - знак на печати занимает 1 позицию;

4) `boolean` - значения "true" и "false" занимают 4 и 5 позиций, соответственно.

Программист имеет возможность при выводе значения задать два формата печати, назовем их *m* и *n*. Эти форматы в общем случае являются выражениями целого типа, чаще всего целыми константами.

При выводе значения любого типа возможно задать количество позиций m (ширину поля) для выводимой величины, например:

```
write(a:m) .
```

Выводимая информация

выравнивается по правому краю поля, то есть если m - избыточно, то лишние позиции заполняются пробелами слева. Если для вывода конкретной информации выделенного поля недостаточно (m - недостаточно), оно автоматически увеличивается до нужного размера (до минимально необходимого числа позиций). Если при выводе вещественного числа ширина поля меньше 8, то она автоматически увеличивается до 8.

Для вещественной величины возможно указать еще число знаков n после десятичной точки, которые необходимо отобразить на экране. Например:

```
write(a:m:n)
```

Если формат

n используется, то вещественное число представляется в форме с фиксированной точкой, в противном случае - с плавающей точкой.


```

(* Анализ бесформатного и форматного вывода *)
var a,b:real; (* Вещественные переменные *)
    i,m:integer; (* Целые переменные *)
    d,l:boolean; (* Логические переменные *)
    c :char; (* Знаковые переменные *)
begin
  a:=5.2; b:=10.375; i:=100; m:=-5; d:=false; l:=true; c:='*';
  writeln('Бесформатный вывод');
  writeln('a=',a,' b=',b);
  writeln('i=',i,' m=',m,' d=',d,' l=',l,' c=',c);
  writeln('Форматный вывод');
  writeln('a1=',a:3:1,' a2=',a:6:1,' a3=',a:3,' b=',b:5:1);
  writeln('i=',i:1,' m=',m:4,' d=',d:7,' l=',l:2,' c=',c:3);
  (* a1 - достаточный размер поля; a2 - избыточный размер поля;
    a3 - недостаточный размер поля для экспоненциального представления,
    автоматически увеличивается до 8 позиций;
    b - недостаточный размер поля для 3 знаков после запятой,
    при выводе будет 1 знак после запятой;
    i,l - недостаточный размер поля, поле автоматически увеличивается
    до нужного размера;
    m,d,c- избыточный размер поля, выводимая информация выравнивается
    по правому краю поля *)
end. (* Конец программы *)

```

Результаты работы программы на экране будут выглядеть следующим образом.

Бесформатный вывод

a= 5.2000000000E+00 b= 1.0375000000E+01

i=100 m=-5 d=FALSE l=TRUE c=*

Форматный вывод

a1=5.2 a2= 5.2 a3= 5.2E+00 b= 10.4

i=100 m= -5 d= FALSE l=TRUE c= *

1.4 ЛАБОРАТОРНАЯ РАБОТА #1 "ЛИНЕЙНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС"

Линейным называется вычислительный процесс, структурная схема которого не содержит разветвлений. В алгоритме линейной структуры все действия выполняются последовательно одно за другим. Для реализации алгоритмов линейной структуры обычно используются операторы ввода-вывода и оператор присваивания.

1.4.1 Пример выполнения лабораторной работы по теме "Линейный вычислительный процесс"

Задание

Дана функциональная зависимость $y=f(x)$ в следующем виде:

$$y(x)=\begin{cases} \frac{\sqrt[3]{x^2 + \pi} + \sin(x)}{1.423117 * \ln(x + \pi)}, & 1 < x \leq 2 \\ e^x + \frac{1.42317}{x}, & 2 < x < 3 \end{cases};$$

Написать программу для вычисления значения y сначала по первой формуле (для $1 < x < 2$), а затем по второй (для $2 < x < 3$). Вычисление y по сложным формулам выполнить по частям с печатью промежуточных результатов (вычислить значение функций, входящих в формулу, затем вычислить значение y через эти промежуточные результаты и распечатать промежуточные результаты и конечное значение y).

Для каждой формулы задать аргумент из указанного интервала и сделать все расчеты на микрокалькуляторе. Полученные данные использовать в качестве теста при проверке работоспособности программы.

Математическая постановка задачи

Математическая постановка не требуется, так как задача уже сформулирована в виде математических формул.

Перед составлением алгоритма необходимо проверить, все ли функции в формулах можно вычислить с помощью стандартных функций Паскаля. Для вычисления

$$\sqrt[3]{x^2 + \pi} = (x^2 + \pi)^{\frac{1}{3}}$$

необходимо воспользоваться соотношением

$$x^n = e^{n \cdot \ln x} \text{ для } x > 0.$$

Получим в итоге:

$$y(x) = \begin{cases} \frac{e^{\frac{1}{3} + \ln(x^2 + \pi)} + \sin(x)}{1.423117 * \ln(x + \pi)}, & 1 < x \leq 2 \\ e^x + \frac{1.42317}{x}, & 2 < x < 3 \end{cases};$$

Обозначения

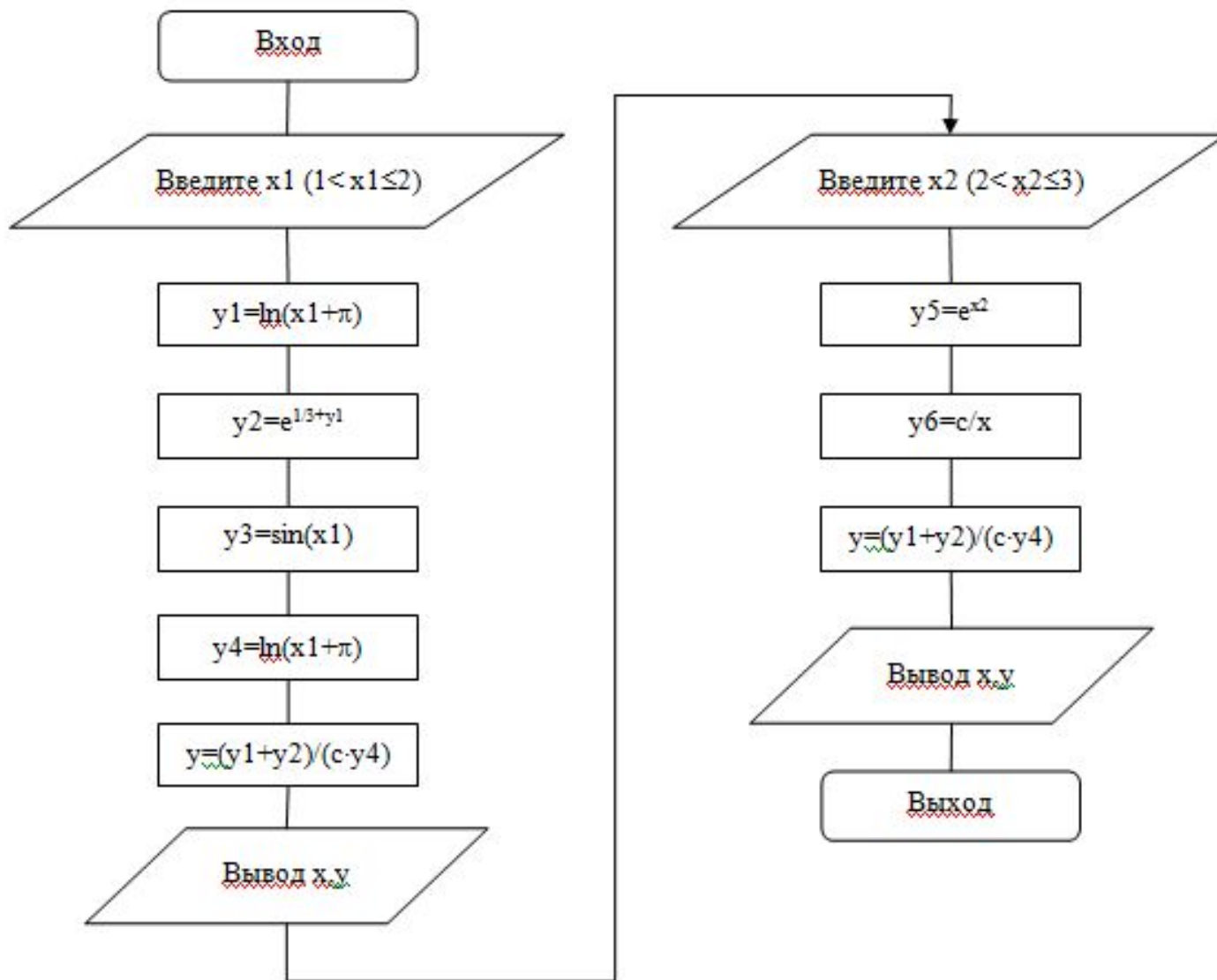
Дано: x - аргумент (real).

Результат: y - значение функции (real).

Промежуточные данные: y_1, y_2, y_3, y_4 - промежуточные результаты (real).

Константы: $\pi = 3.141593$ - константа π ,

$c = 1.423117$.



Программа

```
(* Вычисление  $y=f(x)$  для заданного  $x$  *)
const c =1.412117;
      pi=3.141593;
var x,y:real;
      y1,y2,y3,y4:real;
begin
  writeln('Вычисление  $y=f(x)$  по заданному аргументу  $x$ ');
  (* Вычисление  $y$  по первой формуле для  $1 < x \leq 2$  *);
  write('Введите значение  $x$ , причем  $1 < x \leq 2$ : ');
  readln(x);
  y1:=ln(x*x+pi);
  y2:=exp(1/3 + y1);
  y3:=sin(x);
  y4:=ln(x+pi);
  y:=(y2+y3)/(c+y4);
  writeln(' Для  $x=',x,' y=',y);
  (* Вычисление  $y$  по второй формуле для  $2 < x < 3$  *)
  write('Введите значение  $x$ , причем  $2 < x < 3$ : ');
  read(x);
  y1:=exp(x);
  y2:= c/x;
  y:=y1+y2
  writeln(' Для  $x=',x,' y=',y);
end.$$ 
```


Тесты

| | | | |
|-------|---------|---|------------------------------|
| x=1.5 | y1= ... | } | Конкретные числовые значения |
| | y2= ... | | |
| | y3= ... | | |
| | y4= ... | | |
| | y= ... | | |
| x=2.5 | y= ... | | |

Замечание

Поэтапное вычисление по сложным формулам с получением промежуточных результатов используется в следующих случаях:

- 1) формула длинная и сложная; тогда вычисление по частям делает ее нагляднее, при этом уменьшается вероятность появления ошибок при записи формулы;
- 2) в формуле есть одинаковые повторяющиеся фрагменты; чтобы не записывать их много раз, проще один раз записать этот фрагмент и обозначить его промежуточной переменной, а затем использовать уже эту переменную.