|  |  |
| --- | --- |
|  | ООО «3В Сервис»  РФ, 115191, Москва, Гамсоновский пер., д.2, стр.1 телефон/факс: +7 (495) 221-22-53 [www.3v-services.com](http://www.3v-services.com/) |

**Утверждаю**

Генеральный директор

ООО «ЗВ Сервис»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Петухов В.Н.

****

**Среда динамического моделирования технических систем SimInTech™**

**Описание языка програмирования**

# Реферат

Отчет на 112 стр., 1 рис., 22 таблицы.

БИБЛИОТЕКА ФУНКЦИЙ, ОПЕРАЦИИ, ПРОГРАММИРОВАНИЕ, СИНТАКСИС, СЕМАНТИКА, ФУНКЦИИ, ЯЗЫК ПРОГРАММИРОВАНИЯ.

В документе приводится описание языка программирования, встроенного в среду SimInTech. Описываются общие характеристики языка программирования, синтаксис и семантика базовых и составных элементов языка, библиотека встроенных функций.

Документ составлен в соответствии с требованиями ГОСТ 19.506-79 «Описание языка. Требования к содержанию и оформлению».

# Содержание

[Реферат 2](#_Toc454536820)

[Содержание 3](#_Toc454536821)

[Обозначения и сокращения 6](#_Toc454536822)

[1 Общие сведения 7](#_Toc454536823)

[2 Элементы языка 12](#_Toc454536824)

[2.1 Константы 12](#_Toc454536825)

[2.2 Переменные 13](#_Toc454536826)

[2.3 Операторы 16](#_Toc454536827)

[2.4 Ключевые слова 21](#_Toc454536828)

[2.4.1 Декларации переменных 21](#_Toc454536829)

[2.4.2 Секции 22](#_Toc454536830)

[2.4.3 Подпрограммы 23](#_Toc454536831)

[2.4.4 Циклы 24](#_Toc454536832)

[2.4.5 Условия 25](#_Toc454536833)

[2.4.6 Текстовый препроцессор 26](#_Toc454536834)

[2.4.7 Специальные элементы 29](#_Toc454536835)

[2.5 Процедуры и функции 32](#_Toc454536836)

[3 Библиотека функций 35](#_Toc454536837)

[3.1 Стандартные 35](#_Toc454536838)

[3.1.1 Функция получения модуля вещественного или комплексного числа или вектора 35](#_Toc454536839)

[3.1.2 Функция выбора аргумента по номеру 36](#_Toc454536840)

[3.1.3 Функция вычисления экспоненты вещественного или комплексного числа. 36](#_Toc454536841)

[3.1.4 Функция вычисления десятичного логарифма вещественного или комплексного числа 37](#_Toc454536842)

[3.1.5 Функция ограничения значения заданным интервалом 38](#_Toc454536843)

[3.1.6 Функция вычисления натурального логарифма вещественного или комплексного числа. 39](#_Toc454536844)

[3.1.7 Функция вычисления максимального значения из двух значений или из значений вектора 39](#_Toc454536845)

[3.1.8 Функция вычисления минимального значения из двух значений или из значений вектора 40](#_Toc454536846)

[3.1.9 Функция вычисления значения линейной функции от заданного аргумента, заданной значениями функции и аргументов 41](#_Toc454536847)

[3.1.10 Функция вычисления корня произвольной степени вещественного или комплексного числа 42](#_Toc454536848)

[3.1.11 Функция вычисления корня квадратного вещественного или комплексного числа 43](#_Toc454536849)

[3.2 Логические операторы 44](#_Toc454536850)

[3.2.1 Оператор побитового логического И. 44](#_Toc454536851)

[3.2.2 Оператор целочисленного деления. 44](#_Toc454536852)

[3.2.3 Оператор получения остатка от целочисленного деления. 45](#_Toc454536853)

[3.2.4 Оператор целочисленного или логического отрицания 46](#_Toc454536854)

[3.2.5 Оператор побитового логического ИЛИ 47](#_Toc454536855)

[3.2.6 Оператор побитового сдвига влево. 48](#_Toc454536856)

[3.2.7 Оператор побитового сдвига вправо. 48](#_Toc454536857)

[3.2.8 Оператор побитового логического ИСКЛЮЧАЮЩЕГО ИЛИ 49](#_Toc454536858)

[3.3 Векторные и матричные 49](#_Toc454536859)

[3.3.1 Первый ненулевого элемент в массиве 49](#_Toc454536860)

[3.3.2 Количество столбцов матрицы или количество элементов массива 50](#_Toc454536861)

[3.3.3 Умножение полиномов 52](#_Toc454536862)

[3.3.4 Деление полиномов 53](#_Toc454536863)

[3.3.5 Вычисление детерминанта матрицы 54](#_Toc454536864)

[3.3.6 Создание диагональной матрицы 54](#_Toc454536865)

[3.3.7 Вычисление собственных чисел вещественной матрицы 55](#_Toc454536866)

[3.3.8 Извлечение элементов из матрицы или вектора по номерам 56](#_Toc454536867)

[3.3.9 Создание единичной матрицы 57](#_Toc454536868)

[3.3.10 Вычисление обратной матрицы 58](#_Toc454536869)

[3.3.11 Сортировка вектора по убыванию значений 59](#_Toc454536870)

[3.3.12 Сортировка векторов по убыванию значений определяющего вектора 60](#_Toc454536871)

[3.3.13 Вычисление вектора арифметической прогрессии 61](#_Toc454536872)

[3.3.14 Вычисление вектора арифметической прогрессии в логарифмической шкале 62](#_Toc454536873)

[3.3.15 Решение системы линейных алгебраических уравнений (СЛАУ) 63](#_Toc454536874)

[3.3.16 Cоздание нулевой матрицы 64](#_Toc454536875)

[3.3.17 Создание матрицы единиц 65](#_Toc454536876)

[3.3.18 Размножение каждого элемента вектора 66](#_Toc454536877)

[3.3.19 Вычисление первой нормы вещественной матрицы 66](#_Toc454536878)

[3.3.20 Вычисление бесконечной нормы вещественной матрицы 67](#_Toc454536879)

[3.3.21 Вычисление характеристического полинома вещественной матрицы 68](#_Toc454536880)

[3.3.22 Сложение полиномов 68](#_Toc454536881)

[3.3.23 Вычисление производной от полинома 69](#_Toc454536882)

[3.3.24 Вычисление корней характеристического полинома вещественной матрицы 70](#_Toc454536883)

[3.3.25 Вычитание полиномов 71](#_Toc454536884)

[3.3.26 Вычисление значения полинома 72](#_Toc454536885)

[3.3.27 Вычисление значения, обратного числу обусловленности матрицы 73](#_Toc454536886)

[3.3.28 Количество строк матрицы 74](#_Toc454536887)

[3.3.29 Установка размера матрицы или вектора 75](#_Toc454536888)

[3.3.30 Сортировка вектора по возрастанию значений 76](#_Toc454536889)

[3.3.31 Сортировка векторов по возрастанию значений определяющего вектора 77](#_Toc454536890)

[3.3.32 Суммирование элементов вещественного или комплексного вектора 78](#_Toc454536891)

[3.3.33 Транспонирование матрицы или вектора 79](#_Toc454536892)

[3.3.34 Создание нулевого вектора 80](#_Toc454536893)

[3.3.35 Создание вектора единиц 80](#_Toc454536894)

[3.3.36 Вычисление значений скалярной функции 81](#_Toc454536895)

[3.4 Комплексные 82](#_Toc454536896)

[3.4.1 Получение аргумента комплексного числа 82](#_Toc454536897)

[3.4.2 Получение комплексного числа 82](#_Toc454536898)

[3.4.3 Получения комплексно-сопряженного числа 83](#_Toc454536899)

[3.4.4 Получение мнимой части комплексного числа 83](#_Toc454536900)

[3.4.5 Получение вещественной части комплексного числа 84](#_Toc454536901)

[3.4.6 Получение комплексного квадратного корня вещественного числа 85](#_Toc454536902)

[3.5 Гиперболические 85](#_Toc454536903)

[3.6 Тригонометрические 85](#_Toc454536904)

[3.7 Интерполяция 86](#_Toc454536905)

[3.8 Специальные 86](#_Toc454536906)

[3.9 Свойства воды и водяного пара 88](#_Toc454536907)

[3.10 Статистические 89](#_Toc454536908)

[3.11 Функции случайных чисел 89](#_Toc454536909)

[3.12 Геометрические 90](#_Toc454536910)

[3.13 Файловые 91](#_Toc454536911)

[3.14 Работа с памятью 92](#_Toc454536912)

[3.15 Строковые 93](#_Toc454536913)

[3.16 Функции времени 97](#_Toc454536914)

[3.17 Графические и системные 99](#_Toc454536915)

[3.18 Функции доступа к БД через ADO 116](#_Toc454536916)

[3.19 Функции прямого доступа к интерфейсу обмена с удаленной системой 116](#_Toc454536917)

# Обозначения и сокращения

|  |  |  |
| --- | --- | --- |
| < > | - | обязательный идентификатор в языковой конструкции |
| { } | - | опциональный идентификатор в языковой конструкции |

# Общие сведения

Язык программирования входит в состав графической оболочки среды SimInTech и предназначен для решения следующих основных задач:

- параметризация исходных данных;

- создание новых типовых блоков со своими оригинальными математическими моделями;

- создание скриптов, описывающих поведение графических объектов в модуле анимации.

Язык программирования – интерпретаторного типа. Однако при запуске задачи на расчет происходит предварительная компиляция программы во внутренний псевдокод, что значительно ускоряет выполнение программы.

Исходный текст программы на встроенном языке может располагаться в следующих местах графической оболочки:

- в окне текстового редактора параметров субмодели (появляется при нажатии на закладку «Скрипт» в схемном окне);

- в окне текстового редактора параметров типового блока «Язык программирования» (появляется при двойном нажатии левой кнопки «мыши» на изображении блока);

- в окне текстового редактора скрипта модуля анимации (появляется при выборе меню «Сервис\Скрипт» окна анимации).

Кроме того, имена переменных, заданных в окне текстового редактора параметров субмодели, а также выражения на языке программирования могут использоваться для вычисления значений свойств типовых блоков (для большинства блоков окно свойств типового блока появляется при двойном нажатии левой кнопки «мыши» на изображении блока).

В качестве примера на рисунке 1.1 приведено окно свойств блока «Инерционно-дифференцирующее звено», где коэффициент усиления вычисляется как выражение от константы «A», заданной в окне параметров главной субмодели, встроенной функции языка программирования «sin» и системной переменной «time». При этом, поскольку переменная «time» (модельное время) является переменной, то коэффициент усиления блока «Апериодика первого порядка» определяется ядром среды SimInTech как переменная, и выражение «A + sin(time)» будет пересчитываться на каждом шаге интегрирования.

В графической оболочке среды SimInTech приняты следующие соглашения в отношении области видимости и порядка расчета объявленных переменных и констант:

1. Если переменная или константа объявлена в блоке «Язык программирования», то она является локальной и может использоваться только в этом блоке.

2. Если переменная или константа объявлена в окне параметров субмодели, то она является глобальной (видимой) для всех блоков и субмоделей, подчиненных данной субмодели.

3. Порядок расчета переменных происходит сверху вниз по иерархической лестнице подчиненности блоков на структурной схеме.

4. Если в субмодели 1 объявлена и рассчитывается переменная «А», а в субмодели 2, подчиненной субмодели 1, эта переменная переобъявлена (например, с помощью ключевого слова «var»), то во всех блоках и субмоделях, подчиненных субмодели 2, будет использоваться значение переменной «А» субмодели 2.

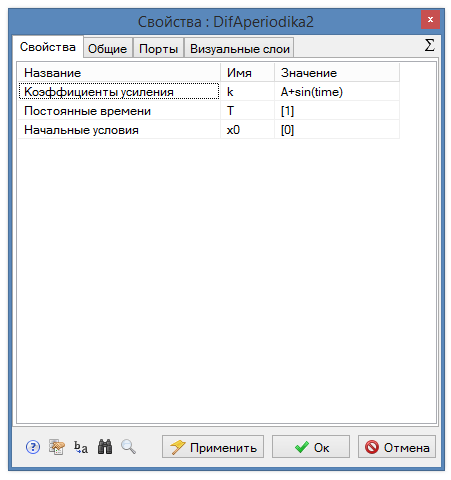


Рисунок 1.1 – Пример параметризации свойства типового блока

Язык программирования предназначен для создания программ, описывающих функционирование типового блока «Язык программирования», а также для задания глобальных параметров и переменных во вкладке «Параметры».

При помощи языка программирования во вкладке «Параметры» можно написать программу, которая может производить по мере расчета манипуляции с объектами схемы, а также запрограммировать видеокадры.

Вкладка «Параметры» есть на каждом уровне вложенности субмоделей и у каждой субмодели может быть записана своя локальная программа (скрипт) в этой вкладке.

Идентификаторы (имена констант, переменных, меток, функций и процедур) могут содержать буквы латинского и русского алфавита, знак подчеркивания «\_»и цифры. Идентификатор должен начинаться с буквы или знака подчеркивания и может иметь произвольную длину. Не допускается использовать в качестве идентификаторов ключевые слова (эти слова автоматически выделяются при вводе полужирным шрифтом). Строчные и прописные буквы по умолчанию в идентификаторах не различаются (t и T – одинаковые идентификаторы). Данные параметр можно изменить в настройках системы.

Программа состоит из деклараций и операторов. Декларации начинаются с ключевых слов и инициализируют соответствующие переменные и константы:

**сonst** – константы;

**var** – алгебраические переменные;

**init** – динамические (дифференциальные) переменные;

**input** – входные переменные блока;

**output** – выходные переменные блока.

После ключевого слова перечисляются через запятую имена переменных либо констант с возможным заданием начальных значений, например:

**const** a=5.3,b=1.2e-4,c=10,y0=a\*b/c;

**var** x1,v=b;

**output** y=y0;

Если начальное значение не задано, то оно принимается равным 0. Разделителем деклараций и операторов является точка с запятой (;).

В декларациях обязательно должны быть описаны входные, выходные и динамические (дифференциальные) переменные. Остальные переменные могут быть заданы автоматически в соответствии с выражением, которое присваивается переменной. Рекомендуется (но не обязательно) помещать декларации в начало программы. Переменную или константу можно использовать только после того, как она описана в декларации или задана в операторе присваивания. Доступны также системные переменные, приведенные в разделе 2.2.

Программа может содержать комментарии, заключенные в фигурные скобки:

*{текст комментария}*

либо в виде:

*//текст комментария до конца строки.*

Приведем несколько простых примеров программ для блока «Язык программирования».

**Пример 1.**

*Выходная переменная блока – максимальное значение из двух входных переменных.*

|  |  |
| --- | --- |
|  | **input** u1,u2;  **if** u1>u2 **then** y=u1 **else** y=u2;  **output** y; |

Можно упростить эту программу, используя функцию **max** языка, тогда оператор вычисления выходной переменной запишется в виде y=**max**(u1,u2);

**Пример 2.**

*Выходная переменная запоминает максимальное значение входной переменной.*

|  |  |
| --- | --- |
|  | **input** u;  **if** time=0 **then** y=u **els**e  **if** goodstep **then** y=**max**(y,u); *//goodstep-флаг «хорошего» шага*  **output** y; |

Значение time=0 соответствует начальному моменту модельного времени. Переменная y практически задает состояние блока. Расчет таких переменных рекомендуется выполнять только на хорошем шаге, когда булевская переменная goodstep равна 1. В противном случае (при goodstep=0) переменная u может быть вычислена, например, на промежуточной итерации решения алгебраических уравнений, в результате чего значение переменной y будет неверным.

**Пример 3.**

*Блок вычисляет время переходного процесса, т.е. время, в течение которого абсолютное значение входной переменной z уменьшается до z0 и в дальнейшем не превышает этой величины.*

|  |  |
| --- | --- |
|  | **const** z0=0.05;  **input** z;  **output** T=0; *//0 – начальное значение*  **if** goodstep **and** (**abs**(z)>z0) **then** T=time; |

**Пример 4.**

*Блок описывает уравнение Ван-дер-Поля.*

|  |  |
| --- | --- |
|  | **const** mu=1e6;  **init** x1=2,x2=0;  x1’=x2; *//дифференциальные уравнения*  x2’=mu\*((1-x1^2)\*x2-x1);  **output** x1,x2; |

Здесь операторы присваивания задают дифференциальные уравнения и определяют новые переменные x1’,x2’ (производные соответствующих переменных), которые, как и другие переменные, могут быть использованы в математических выражениях.

**Пример 5.**

*Счетчик числа поступающих на вход импульсов.*

|  |  |
| --- | --- |
|  | **input** u;  **output** Nimp=0;  **var** u0=1;  **if** goodstep **and** (u0<=0) **and** (u>0) **then**  **begin** Nimp=Nimp+1; u0=u **end**; |

Импульсы считаются по переднему фронту, т.е. моменту модельного времени, когда входная переменная возрастает и становится положительной.

# Элементы языка

В описании элементов языка символами «<», «>» помечены обязательные идентификаторы. Символами «{», «}» помечены дополнительные (опциональные) идентификаторы.

## Константы

Константы делятся на:

* числовые;
* строковые;
* специальные встроенные.

Числовые константы задаются в языке программирования стандартным образом, аналогично другим языкам высокого уровня. Формат задания чисел:

{<знак>}<целая часть>{.<десятичная часть>}{e<показатель степени>}{i}.

Десятичный разделитель – точка. Символы i j I J непосредственно после числа означают, что число является мнимым, например мнимая единица обозначается 1i.

Примеры числовых констант:

10.5, –10.5e–7, 2.3+1.1e–5i.

Строковые константы задаются в виде "строка символов".

Константы, как и переменные, могут быть векторными и матричными. Вектор-строка задается как набор констант, разделенных между собой запятыми либо пробелами и заключенных в квадратные скобки. Настоятельно рекомендуется использовать в качестве разделителя запятую, например [2,–1,3] вместо [2 –1 3], поскольку в последнем случае вектор будет воспринят как [(2–1) 3]=[1 3].

Матрица задается как набор векторных констант (строк), разделенных между собой запятыми и заключенных в квадратные скобки, например [[2,–1,3],[4,5,6]] – матрица размером 2×3.

Именованные константы задаются с помощью ключевого слова **const**, после которого перечисляются (через запятую) имена констант с присвоенными им значениями, например:

**const** c=1e-3, v=[c,1,3.2], M=[v,[1.5,2,2.2]];

Специальные встроенные константы:

e = 2.7182818284590452353602874713527 - основание натурального логарифма

pi = 3.14159… - число π.

## Переменные

Переменная – это именованная ячейка памяти, в которую можно записать или из которой можно прочитать значение. Переменные могут быть созданы при помощи ключевых слов **var**, **const**, **init**, **output**, **input** или автоматически, в соответствии с выражением, которое присваивается переменной, а также получены из Глобальных Параметров субмодели. Переменные могут иметь один из перечисленных ниже стандартных типов.

Идентификатор типа Описание типа данных

**integer** Целое значащее 32-битное число

**double** Действительное 64-битное число

**complex** Комплексное 128-битное число. Имеет формат вывода a+{-}bi,

где a – действительная часть, b – комплексная часть.

Можно задать комплексное число также в виде (a,b)

**intarray** Массив (вектор-строка) целых чисел

**array** Массив (вектор-строка) действительных чисел

**carray** Массив (вектор-строка) комплексных чисел

**matrix** Матрица действительных чисел

**cmatrix** Матрица комплексных чисел

**boolean** Двоичное 1-битное значение, может иметь значение 0 или 1

**color** Цвет – эквивалентен типу **integer**

**point** Геометрическая точка – эквивалентна комплексному числу,

имеет формат вывода (a,b), где a – абсцисса, b – ордината

**string** Строка символов

***Примечание.*** Далее под вектором понимается массив (вектор-строка) чисел, а вектор-столбец – это матрица, состоящая из нескольких строк и одного столбца.

Кроме стандартных типов переменных могут быть переменные сложного типа – *записи*.

*Запись* декларируется при указании типа данных переменной в соответствии со следующим форматом:

<имя поля 1>{:<тип поля 1>}{=<начальное значение поля 1>}{…}

Пример:

|  |  |
| --- | --- |
|  | **var** newrec : (x = 0.0 , y = 0.0); |

***Примечание.*** Поля внутри декларации сложного типа определяются аналогично ключевому слову **var**.

Доступ к полю сложной переменной производится при помощи оператора « **->** » в соответствии со следующим форматом:

<имя записи> **->** <имя поля записи>

Пример:

|  |  |
| --- | --- |
|  | A = newrec**->**x; |

В блоке "Язык программирования" доступны следующие системные переменные:

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Назначение |
| **time** | Вещественное (double) | Модельное время |
| **stepsize** | Вещественное (double) | Шаг интегрирования |
| **goodstep** | Двоичное (Boolean) | Флаг «хорошего» шага |
| **getderi** | Двоичное (Boolean) | Флаг вычисления значений производных |
| **setstepflag** | Двоичное (Boolean) | Флаг принудительного присвоения шага |
| **newstepvalue** | Вещественное (double) | Принудительное значение шага при **setstepflag = 1** |

Переменная goodstep принимает значение 1, если вычисления производятся в конечной точке шага интегрирования, и при этом удовлетворяется условие заданной точности; во всех других случаях переменная goodstep принимает значение 0. Переменная getderi принимает значение 1 при вычислении матрицы Якоби посредством численного дифференцирования; во всех других случаях переменная goodstep принимает значение 0.

Специальные переменные доступные в модуле анимации:

|  |  |
| --- | --- |
| Имя | Назначение и пример |
| **GROUPCOLOR** | Цвет фона группы |
| **GLOBALCOLOR** | Цвет фона главного окна |
| **PAINTSTEP** | Флаг, выставляемый при перерисовке изображения окна. |
| **CONTAINER\_NAME** | Имя графического контейнера, в котором написан скрипт. |
| **SCALEX** | К-т масштабирования контейнера по оси X |
| **SCALEY** | К-т масштабирования контейнера по оси Y |
| **KLINE** | К-т масштабирования линий внутри контейнера |
| **CHANGE** | Флаг принудительной перерисовки графического контейнера. |
| **PARENTOBJECT** | Ссылка на объект-владелец контейнера (для того чтобы изнутри контейнера осуществлять операции с объектом) |
| **old\_project\_name** | Имя предыдущего файла, загруженного в окно |
| **old\_link\_id** | Имя последнего идентификатора ссылки, загруженного в окно |
| **last\_active\_screen\_id** | screen\_id последнего активного окна |
| **screen\_id** | Идентификатор окна редактора – произвольная строка, присваиваемая в скрипте |
| **last\_active\_project\_id** | Ссылка окна последнего активного проекта |
| **SYSTEM\_DB\_ROOT** | Путь хранения различных файлов данных, установленный для графической оболочки. |
| **DYNAMIC\_PROPERTY\_ITEMS** | Переменная для указания через скрипт подсказки выбора (для скриптов внутри свойств блока) |
| **last\_active\_hwnd** | Указатель на последнее активное окно (хэндл) |
| **restartstep** | Шаг сохранения рестартов |
| **time** | Модельное время |
| **submodel** | Указатель на блок-родитель субмодели |

Переменные могут задаваться (декларироваться) при помощи ключевых слов деклараций: **var**, **const, input, output, init,** которые будут рассмотрены в соответствующем разделе.

Язык программирования допускает использование доступа к переменным при помощи ссылок. Ссылка – это вещественное 64-битное число, первые 4 байта которого представляют собой адрес области данных, а вторые 4 байта – идентификатор типа данных ссылки, который используется для контроля доступа.

Для получения ссылки переменной, выделенной статически, используется оператор @.

**@**<переменная или функция>

Для получения доступа к переменной по ссылке используются функция типизированного доступа «~», dispose и «new~», которые рассмотрены в разделе РАБОТА С ПАМЯТЬЮ.

Примечание: неправильное использование ссылок может привести к серьёзным сбоям в программе. Не используйте их без необходимости.

## Операторы

Каждая переменная или константа характеризуется своим значением и принадлежит к какому-либо типу данных. С помощью операторов и круглых скобок из них можно составлять выражения, которые фактически представляют собой правила получения новых значений. В общем случае выражение состоит из нескольких элементов (операндов) и знаков операций (операторов), а тип его значения определяется типом операндов и видом применяемых к ним операций. Если операции имеют равный приоритет, то они будут выполняться последовательно, слева направо. Если приоритет операции выше, то она будет выполнена в первую очередь, независимо от расположения знака операции. В первую очередь вычисляются также выражения, заключенные в скобки, после чего они рассматриваются как операнды. В выражении (x>0)**and**(y>0) скобки необходимы, поскольку операции отношения имеют самый низкий приоритет.

Унарные операторы:

|  |  |  |
| --- | --- | --- |
| Оператор | Семантика | Назначение и пример |
| **-** | - <выражение> | Оператор отрицания выражения.  **Пример:**  -1 |
| **not** | **not** <логическое или целочисленное выражение> | Оператор целочисленного или логического отрицания.  **Пример:**  **not** (A > 10) |
| + | + <выражение> | Унарный плюс – игнорируется. |
| @ | @<переменная> | Вычисляет ссылку данных переменной. Ссылка – это 32 битное целое число, соответствующее адресу переменной в памяти. |

В арифметических операциях операндами могут быть вещественные, комплексные и целые числа, массивы, матрицы, а также выражения соответствующих типов. Операции над матрицами определены в соответствии с правилами линейной алгебры, а операции над массивами выполняются поэлементно (в последнем случае размеры массивов должны совпадать). Массив воспринимается как вектор-строка. Вектор-столбец задается как матрица размером n×1, например A=[[1],[3],[2]]; либо как транспонированный массив: A=**transp**([1,3,2]);

|  |  |  |
| --- | --- | --- |
| Оператор | Семантика | Назначение и пример |
| **+** | <слагаемое 1>+<слагаемое 2> | Сложение двух выражений. Для строк сложение соответствует конкатенации. Матрицы и вектора складываются поэлементно.  **Пример:**  1 + 10; |
| **-** | <уменьшаемое>-<вычитаемое> | Вычитание из первого выражения второго. Для строк не определено. Матрицы и вектора вычитаются поэлементно.  **Пример:**  10 + 5  [1,1] + [5,7] |
| **\*** | <множитель 1>\*<множитель 2> | Умножение двух выражений. Для строк не определено. Вектора перемножаются поэлементно. Матрицы перемножаются по правилам умножения матриц.  **Пример:**  A \* B |
| **/** | <делимое>/<делитель> | Прямое деление первого выражения на второй. Вектора делятся поэлементно. Для матриц производится умножение на обратную матрицу.  **Пример:**  3/10  [1,1]/[10,20] |
| **\** | <матрица A>\<матрица B> | Обратное деление матриц. Соответствует операции A-1\*B |
| **^** | <базовое значение>^<степень> | Возведение первого выражение в степень второго. Вектора обрабатываются поэлементно. Матрицы возводятся в целую степень по правилам умножения матриц. Для возведения матрицы в степень поэлементно используйте оператор **.^**  **Пример:**  2^3 |
| **.\*** | <матрица 1>.\*<матрица 2> | Поэлементное перемножение действительных или комплексных матриц. |
| **./** | <матрица 1>./<матрица 2> | Поэлементное деление действительных или комплексных матриц. |
| **.^** | <матрица 1>.^<матрица 2> | Поэлементное возведение в степень двух матриц или матрицы в действительную степень |
| **!** | <целое число>! | Вычисление факториала целого числа |
| **..** | a..b | Вычисление интервала целых чисел от a до b.  **Пример:**  1..4 обозначает [1,2,3,4] |
| **#** | A**#**B | Размножение числа B в вектор размерностью A.  **Пример:**  3#0.1 обозначает [0.1, 0.1, 0.1] |

Целочисленные и логические операторы:

|  |  |  |
| --- | --- | --- |
| Оператор | Семантика | Назначение и пример |
| **or** | <выражение 1> **or** <выражение 2> | Побитовая логическая операция ИЛИ. |
| **and** | <выражение 1> **and** <выражение 2> | Побитовая логическая операция И. |
| **xor** | <выражение 1> **xor** <выражение 2> | Побитовая логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ. |
| **div** | <делимое> **div** <делитель> | Целочисленное деление. |
| **mod** | <делимое> **div** <делитель> | Остаток от целочисленного деления. |
| **shl** | <число> **shl** <количество сдвигов> | Побитовый сдвиг влево. |
| **shr** | <число> **shr** <количество сдвигов> | Побитовый сдвиг вправо. |

Отдельной категорией операторов являются скобки. В языке программирования используется три вида скобок: круглые «()», квадратные «[]» и фигурные «{}». Фигурные скобки позволяют создавать строки комментария в тексте программы.

Способы использования круглых и квадратных скобок показаны в таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| **Оператор** | **Семантика** | **Назначение** | **Пример** |
| **( )** | **(**<выражение>**)** | Математические скобки. Выражение внутри скобок выполняется первым. | 4\***(**10+7**)** |
| **( )** | **(**<действительная часть>, <мнимая часть>**)** | Упаковка действительных чисел (векторов, матриц) в комплексное (вектор, матрицу). Допустимо написание частей комплексного числа через пробел или точку с запятой. | **(**10, 5**)** |
| **[ ]** | **[**<элемент 1> {,<элемент 2>}**]** | Упаковка элементов в массив. Если элементы – действительные или целые числа, то результат – вектор действительных чисел. Если элементы – векторы действительных чисел, результат – матрица действительных чисел. Если элементы – комплексные числа, то результат – вектор комплексных чисел. Если элементы – векторы комплексных чисел, результат – матрица комплексных чисел. Разделителем элементов может быть запятая, точка с запятой или пробел. Если элемент вектора отрицательное число, то данный элемент в векторе должен отделяться от предыдущего запятой. Вектора и матрицы можно присваивать, при этом элементы должны быть переменными. | **Правильно:**  X = [0, -0.23]  **Неправильно:**  X = [0 -0.23]  **Вектор:**  **[**1, 4.5, 7**]** или **[**1 4.5 7**]**  **Матрица:**  **[[**1,2**],[**6,7**]]** или **[[**1 2**];[**6 7**]]**  **Присваивание элементов**  **[**a,b**]** = **[**1, 2**]** |
| **[ ]** | <массив или матрица> **[**<номер элемента или строки> {,<номер столбца>}**]** | Возвращает значение элемента с указанным номером. Нумерация начинается с 1. Тип возвращаемого значения соответствует типу элементов массива. Если для матрицы указывается один индекс, то возвращается вектор-строка матрицы, если 2 – возвращается число. Этот оператор может быть присвоен. Разделителем элементов может быть запятая, точка с запятой или пробел. | **Доступ:**  A=C**[**1,1**]** k=arr**[**i**]**  **Присваивание**:  C**[**1,1**]** = 10  k**[**3**]** = 4 |

***Примечание:*** Cкобки массивов [ ] могут использоваться для декларации действительных массивов и матриц аналогично «Новому блоку». (см. описание ключевого слова **var**).

Еще одним отдельным оператором является присваивание. Присваивание осуществляется при помощи оператора «=» в соответствии со следующим форматом:

<имя переменной или функция доступа переменной> **=** <выражение>

Если переменная не была ранее декларирована, то создаётся новая переменная, тип которой определяется по типу выражения.

Пример:

|  |  |
| --- | --- |
|  | A = 10;  Var M:matrix;  M=[[0,1],[1,0]];  M[1,1] = sin(10); |

## Ключевые слова

### Декларации переменных

Список деклараций языка программирования приведен в таблице:

|  |  |  |
| --- | --- | --- |
| **Ключевое слово** | **Семантика** | **Назначение и пример** |
| **Var** | **var** <имя переменной 1>{:<тип переменной 1>}{=<инициирующее выражение 1>}{**,**<имя переменной 2>{:<тип переменной 2>}{=<инициирующее выражение 2>}}**;** | Декларация новых переменных. Разделитель деклараций – запятая. Тип переменных по умолчанию – действительное число. Если перед именем переменной указано слово **init** (или **dynamic**), то такая переменная будет считаться динамической (см. описание ключевого слова **init**). Для декларации действительных массивов и матриц можно пользоваться такой конструкцией:  X[10] – массив размерностью 10.  M[2,3] – матрица из 2 строк и 3 столбцов.  Пример:  **Var** d:double = 0.5,I:integer,k = [1 5], **init** x = 10, **dynamic** v = 0; |
| **const** | **Const** <аналогично **var**> | Декларация констант. Если константа потом переопределена, она будет воспринята как переменная. Константы отличаются от переменных тем, что они будут рассчитаны только один раз в начале расчёта. Формат задания переменных такой же как и у ключевого слова **var**.  Пример: **Const** c = 1e-3, v = [c,1,3.2], M = [v,[1.5,2,2.2]]; |
| **Input** | **Input** <аналогично **var**>; | Декларация входных переменных блока. Значения из входов блока будут передаваться в эти переменные. Формат задания переменных такой же как и у ключевого слова **var**.  Пример: **Input** a,b,c; |
| **output** | **Output** <аналогично **var**>; | Декларация выходных переменных вычислительного блока. Значения этих переменных передаются на выходы блока. Формат задания переменных такой же как и у ключевого слова **var**.  Пример: **Output** d,e,f; |
| **Init** | **Init** <аналогично **var**> | Декларация динамических переменных. Например:  **Init** x, x1;  По этой команде будут созданы две переменные одинакового типа: переменная состояния **x** и её производная **x’**. Если x – массив переменных состояния, то x'[i] – производная переменной x[i] этого массива. Переменные состояния позволяют решать дифференциальные уравнения в форме Коши вида: X’ = f(X); |

### Секции

Язык программирования позволяет выполнять код на разных этапах моделирования. Для этого предназначен механизм секций. Язык программирования SimInTech реализует несколько секций.

Формат секции инициализации имеет следующий вид:

|  |  |
| --- | --- |
|  | **initialization**  <операция 1>;  <операция 2>;  **end**; |

Операция внутри секции будут выполнены только один раз при запуске вычислений. В пределах блока можно задавать несколько секций инициализации. Операции вне этих секций выполняются на каждом шаге.

Формат секции финализации имеет следующий вид:

|  |  |
| --- | --- |
|  | **finalization**  <операция 1>;  <операция 2>;  **end**; |

Операции выполняются после того, как все остальные вычисления будут закончены. Операции выполняются однократно при завершении моделирования (после завершения всех шагов - либо при достижении конечного времени интегрирования, либо при нажатии пользователем кнопки стоп). В пределах блока можно задавать несколько секций.

Формат секции с локальными переменными имеет следующий вид:

|  |  |
| --- | --- |
|  | **local**  <операция 1>;  <операция 2>;  **end**; |

Задается секция, внутри которой не доступны для записи внешние переменные, т.е. переменные, которые декларированы в параметрах вышестоящей по уровню субмодели или в общем списке сигналов проекта. При присваивании внешней переменной внутри этой секции автоматически создаётся внутренняя переменная с тем же именем.

Формат секции выполнения в процессе компиляции имеет следующий вид:

|  |  |
| --- | --- |
|  | **beforecompile**  <операция 1>;  <операция 2>;  **end**; |

Задание секции выполнения операций в процессе компиляции текста скрипта. Операции внутри секции будут исполнены сразу, как только заканчивается данная секция. Данная секция может использоваться, например, для динамического ввода нужных сигналов, которые используются далее.

Пример:

|  |  |
| --- | --- |
|  | **beforecompile** //секция выполнения в процессе компиляции    **if not signalexist**("gant\_s") **then**  **addsignaltolist**(13,0,"gant\_s","gant\_s","[[0]]",2); **if not signalexist**("gant\_tend") **then**  **addsignaltolist**(13,0,"gant\_tend","gant\_tend","[[0]]",2); **if not signalexist**("gant\_t0") **then**  **addsignaltolist**(13,0,"gant\_t0","gant\_t0","[[0]]",2);  **end**;  gant\_s = [1,2,3]; |

В примере происходит динамическое добавление новых сигналов в список сигналов проекта с дальнейшим использованием созданных сигналов в том же скрипте.

### Подпрограммы

В языке программирования предусмотрен механизм подпрограмм. Для выделения подпрограммы задаются так называемые операторные скобки:

**begin**

<операция 1>;

…

<операция n>;

**end**;

Секция операторных скобок объединяет несколько операторов в один. Также операторные скобки могут быть вложенными.

Пример:

|  |  |
| --- | --- |
|  | **var** i,id;  **if getprop**(id, "tag") = 1 **then**  **begin**  //изменим значения свойств  **setprop**(id,"tbl\_name",submodel.tbl\_name);  **setprop**(id,"ob\_name",submodel.ob\_name);  **end**;  i = i + 1; |

В примере при помощи операторных скобок объединяются операции, выполняемые по условию.

Для выхода из текущего блока подпрограммы предусмотрен оператор exit.

**exit;**

Если оператор выполняется внутри функции, то происходит выход из функции. Если внутри субмодели, то завершается выполнение скрипта на данном шаге субмодели.

Пример:

|  |  |
| --- | --- |
|  | Label1: i=i+1;  **if** i<10 **then goto** Label1;  **if** i>10 **then exit**; |

Подпрограмма также может быть выполнена в виде пользовательской функции, задание которой рассмотрено в разделе ПРОЦЕДУРЫ И ФУНКЦИИ.

### Циклы

Язык программирования поддерживает три типа циклов:

- цикл с предусловием;

- цикл с постусловием;

- конечный цикл.

Цикл с предусловием обеспечивает выполнение операций внутри цикла до тех пор, пока выполняется условие цикла и имеет следующий формат:

**while** <условие цикла> **do** <операция выполняемая в цикле>;

Пример:

|  |  |
| --- | --- |
|  | **var** i = 1;    **while** i <= **10 do**  **begin**  i = i + 1;  **end**; |

В результате выполнения примера переменной i будет присвоено значение 11.

Цикл с постусловием обеспечивает выполнение операций, заключённых между ключевыми словами «repeat» и «until» до тех пор, пока не будет выполнено условие выхода из цикла, и имеет следующий формат:

**repeat**

<операция 1>;

<операция n>;

**until** <условие выхода>;

Пример:

|  |  |
| --- | --- |
|  | **var** i = 1;    **repeat**  i= i+1;  **until** i=10; |

В результате выполнения примера переменной i будет присвоено значение 11.

В конечном цикле операции выполняются, пока значение переменной цикла не достигнет конечного значения. На каждом шаге значение переменной цикла изменяется на величину, равную указанному шагу. Если шаг не указан, то он равен 1 или -1 в зависимости от того, больше конечное значение начального или меньше. Формат конечного цикла имеет следующий вид:

**for** (<операция присваивания переменной цикла>,<конечное значение>{,<шаг переменной цикла>}) <операция цикла>;

Пример:

|  |  |
| --- | --- |
|  | **var** s=0;  **for** (i=1,10)  s = s + i^2; |

В результате выполнения примера переменной i будет присвоено значение 11, переменной s присвоено значение 385

### Условия

В языке программирования реализованы операторы условного и безусловного переходов. Условный переход - это выполнение операций в зависимости от условия. Условием должно быть логическое или целочисленное выражение. При выполнении условия выполняется операция после ключевого слова then, в противном случае выполняется операция после ключевого слова else. Допустимо задание сложных условий.

Формат операции условного перехода имеет следующий вид:

**if** <условие или логическое выражение>

**then** <операция при выполнении условия>

**else** <операция при невыполнении условия>;

Пример:

|  |  |
| --- | --- |
|  | **var** k = 5;  **if** k>0 **then** k=0  **else**  **begin**  k=-1; i=0  **end**; |

Безусловный переход осуществляется при помощи оператора goto. Формат оператора:

<имя метки>:<операторы назначения>;

**goto** <имя метки>;

Это означает безусловный переход к операции, помеченной меткой. Безусловные переходы могут быть осуществлены только в пределах одного программного блока (функции или секции алгоритма).

Пример:

|  |  |
| --- | --- |
|  | Label1: i=i+1;  **if** i<10 **then goto** Label1;; |

### Текстовый препроцессор

Задание секции текстового процессора осуществляется командой formattext, имеющей следующий формат:

**formattext**

<переменная 1> = <{текст 1}>;

<переменная 2> = <{текст 2}>;

**end;**

Секция выполняется перед началом инициализации (как и операторы условной компиляции). Текст, заключенный в фигурные скобки, заменяется на значение свойства объекта и присваивается переменной.

Пример:

|  |  |
| --- | --- |
|  | **formattext**  **ifdef** ISST1  SET1 = {Name1}\_L1\_SET;  **endif**  **end**; |

В примере формируется переменная *SET1* из переменной *Name1* и постфикса *\_L1\_SET*.

Еще одной возможностью языка программирования является компиляция программы частично в соответствии с заданным условием. Условие компиляции задаётся при помощи ключевого слова **define.**

**define** <Имя условия>

Пример:

|  |  |
| --- | --- |
|  | **define** f\_1;  **ifdef** f\_1   Y = sin(X);  **elseif**  Y = cos(X);  **endif** |

В примере в результате определения условия f\_1 будет скомпилирована строка Y = sin(X);. Сброс же условия компиляции с указанным именем происходит с помощью команды undefine:

**undefine** <Имя условия>

Пример:

|  |  |
| --- | --- |
|  | **undefine** f\_1;  **ifndef** f\_1   Y = sin(X);  **elseif**  Y = cos(X);  **endif** |

В примере в результате определения условия f\_1 будет скомпилирована строка Y=sin(X);

Условная компиляция может осуществляться секциями. Для задания секции условной компиляции используется оператор ifdef:

**ifdef** <Имя условия компиляции>

...

**elseif**

...

**endif**

Если условие задано ранее оператором **define**, то будет скомпилирован текст непосредственно после **ifdef**, а если условие не задано, то будет скомпилирован текст после **elseif**.

Пример:

|  |  |
| --- | --- |
|  | **define** f\_1;  **ifdef** f\_1   Y = sin(X);  **elseif**  Y = cos(X);  **endif** |

В примере в результате определения условия f\_1 будет скомпилирована строка Y = sin(X); Противоположным является оператор ifndef:

**ifndef** <Имя условия компиляции>

...

**elseif**

...

**endif**

Если условие задано ранее оператором **define**, то компилируется текст после **elseif**, а если не задано – то после **ifndef**.

Пример:

|  |  |
| --- | --- |
|  | **define** f\_1;  **ifndef** f\_1   Y = sin(X);  **elseif**  Y = cos(X);  **endif** |

В примере в результате определения условия f\_1 будет скомпилирована строка Y=cos(X);

Язык программирования позволяет создавать библиотеки функций в виде текстовых файлов. Ключевое слово **include** позволяет загрузить фрагмент программы из текстового файла.

**include** "<Имя файла>";

Также из файла можно загрузить произвольный код (скрипт). Например, для комплекта видеокадров, общий функционал можно вынести в файл, и с каждого кадра подгружать общий код.

Пример:

|  |  |
| --- | --- |
|  | **include** "filename.txt"; |

Данный пример позволяет использовать в программе функции, содержащиеся в файле filename.txt.

### Специальные элементы

Язык программирования позволяет загружать библиотеки расширения программ с помощью команды loadmodule:

**loadmodule** "<Имя файла>";

Ключевое слово loadmodule позволяет динамически загрузить плагин (DLL-библиотеку, специально сделанную заранее по определенным правилам) для системы. Функционал плагина может быть произвольный.

Пример:

|  |  |
| --- | --- |
|  | **loadmodule** "filename"; |

Интерпретация текстовых выражений в процессе выполнения программы является еще одним специальным элементом языка SimInTech и доступна по команде eval:

**eval**(<строковая переменная>);

При помощи ключевого слова eval язык программирования позволяет интерпретировать текстовые выражения непосредственно при выполнении программы.

Пример:

|  |  |
| --- | --- |
|  | **beforecompile**    //Эта секция выполняется ДО компиляции остального скрипта    **if** **signalexist**(Name1+"\_L1\_SET") **then eval**("define ISST1") **else eval**("undefine ISST1");  **if** **signalexist**(Name1+"\_H1\_SET") **then eval**("define ISST4") **else eval**("undefine ISST4");    //определяем флаги наличия уставок  //если какой-то уставки нет, то выключаем флаг !!! иначе - //включаем  **if signalexist**("AS\_"+Name1+"\_L1\_ON") **then eval**("define LAS11") **else eval**("undefine LAS11");  **if signalexist**("AS\_"+Name1+"\_H1\_ON") **then eval**("define HAS11") **else eval**("undefine HAS11");    **if signalexist**("PS\_"+Name1+"\_L1\_ON") **then eval**("define LPS11") **else eval**("undefine LPS11");  if signalexist("PS\_"+Name1+"\_H1\_ON") **then eval**("define HPS11") **else eval**("undefine HPS11");    //теперь, если флаг есть - то уставка есть, иначе - не делаем // ничего с этой уставкой  **end**; |

В примере определена секция скрипта, выполняемая до компиляции основного скрипта. В секции определяется наличия уставок для сигналов, если какой-то уставки нет, то выключаем флаг наличия уставки, иначе – включаем. При помощи ключевого слова **eval** определяются текстовые выражения, которые интерпретируются в момент выполнения скрипта.

В языке программирования есть возможность параллельного выполнения задач в разных потоках с помощью команды process:

**process**

<задача 1>;

<задача n>;

**end;**

Объект не возвращает управление пока все задачи, выполняемые параллельно, не завершатся. Задачи задаются в виде обычных пользовательских функций без параметров, но для них будут недоступны операции записи в глобальные переменные. Внутри задач можно организовывать аналогичным образом подзадачи, которые будут выполняться параллельно.

Пример:

|  |  |
| --- | --- |
|  | **var** connected1: **boolean**,  connected2: **boolean**;    //Функция обработки соединения 1  **procedure** DoProcessConnection1   …  **end**;    //Функция обработки соединения 2  **procedure** DoProcessConnection2  …  **end**;  //Синхронная обработка соединений в отдельных потоках  **process**  DoProcessConnection1; //Подключение канала 1  DoProcessConnection2; //Подключение канала 2  **end**; //конец секции |

В примере определяются пользовательские функции обработки соединения, после этого организуется параллельная обработка соединений в разных потоках. Секция не завершится, пока задачи подключения каналов не завершатся.

В случае, если необходимо выполнять задачи без ожидания выполнения каждой, то необходимо использовать команду asyncprocess:

**asyncprocess**

<задача 1>;

<задача n>;

**end**;

Задачи задаются в виде обычных пользовательских функций без параметров, но для них будут недоступны операции записи в глобальные переменные. Внутри задач можно организовывать аналогичным образом подзадачи, которые будут выполняться параллельно.

Пример:

|  |  |
| --- | --- |
|  | **var** connected1: **boolean**,  connected2: **boolean**;    //Функция обработки соединения 1  **procedure** DoProcessConnection1   …  **end**;    //Функция обработки соединения 2  **procedure** DoProcessConnection2  …  **end**;  //Асинхронная обработка соединений в отдельных потоках  **asyncprocess**  DoProcessConnection1; //Подключение канала 1  DoProcessConnection2; //Подключение канала 2  **end**; //конец секции асинхронной обработки |

В примере определяются пользовательские функции обработки соединения, после этого организуется параллельная обработка соединений в разных потоках.

## Процедуры и функции

Обращение к функциям производится согласно следующим правилам

<имя функции>{(<параметр 1>{,<параметр 2>…})}

Функция может не иметь параметров. Разделителем параметров функции может быть запятая, точка с запятой или пробел.

Пример:

**sin**(10 + 5) - с параметрами

**Random** - без параметров

Некоторые функции могут быть присвоены, такие функции называются *функциями доступа*.

Пример:

**real**(C)=10 - присваивание действительной части комплексного числа.

Функции могут иметь несколько разных модификаций в зависимости от типов параметров. Могут быть функции, в которых через параметры возвращаются результаты. В этом случае типы параметров должны совпадать, а выражение, подставляемое в параметр, должно допускать присваивание.

Язык программирования позволяет пользователю создавать собственные функции на базе стандартных операций и функций, аналогично языкам программирования высокого уровня.

Пользовательские функции задаются при помощи ключевых слов:

**function**

…

**end**;

или

**procedure**

…

**end**;

Формат задания пользовательской функции следующий:

**function** <Имя функции>(<имя входной переменной 1>{:<тип входной переменной 1>} {=<инициирующее выражение 1>}{,out <имя выходной переменной> {:<тип выходной переменной>} {=<инициирующее выражение>}}) {:<тип выходного значения>}

<операции выполняемые привызове функции>;

**end**;

Значение функции может быть не присвоено. Выходные переменные декларируются словом out – в такую переменную функция записывает значение. Если тип значения функции не указан, то он определяется автоматически по типу выражения, где присваивается выходное значение функции. Вместо function можно использовать procedure. Если типы параметров функции не указаны, то при создании кода функции типы принимаются как у операндов при соответствующем вызове.

Пример: функция определена следующим образом:

|  |  |
| --- | --- |
|  | **Function** PolyF(A,B)  PolyF = A\*A + B\*B;  **End**; |

Если при вызове операнды A и B будут действительными числами, то результат тоже будет действительным числом. Если A и B будут векторами – то результат будет вектором.

По ходу текста одна и та же функция может быть по-разному определена несколько раз, а также переопределена под другим именем в соответствии со следующим форматом:

**function** <имя функции>=<имя ранее определённой функции>

Пример:

|  |  |
| --- | --- |
|  | **function** SumQ(A,out R:integer):complex  //Присвавание значений функции  SumQ = **sqr**(A);  //Присваивание выходной переменной  R = round(real(SumQ));  **end**;  //Переопределение SumQ как FSum  **function** Fsum=SumQ; |

# Библиотека функций

В разделах 3.1…3.19 приведено базовое описание встроенных функций языка программирования SimInTech. Более подробная информация содержится в справочной системе SimInTech.

## Стандартные

### Функция получения модуля вещественного или комплексного числа или вектора

**Синтаксис:**

*Y* **= abs***(X);*

**Аргументы:**

*Х* – входное значение или входной массив, содержащий элементы вектора.

**Описание:**

*abs(X)* – функция получения модуля действительного или комплексного числа или вектора.

Входное значение или массив *X* может задаваться:

 как переменная типа число или массив чисел, определенная ранее:

*Y* = **abs**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*Y* = **abs**([*x1,x2,x3,x4*]);

 как постоянный массив:

*Y* = **abs**([1,2,3]);

 как постоянное вещественное число:

*Y* = **abs**(-3);

 как постоянное комплексное число:

*Y* = **abs**(3+4i);

Входные значения могут быть как вещественными, так и комплексными числами.

**Результат:**

*Y* – модуль входного значения или выходной массив, элементы которого являются модулями элементов входного массива.

**Пример 1:**

*Модуль вещественного числа*

|  |  |
| --- | --- |
|  | Y = **abs**(-3); |

В результате переменной *Y* будет присвоено значение +3.

**Пример 2:**

*Модуль комплексного числа*

|  |  |
| --- | --- |
|  | Y = **abs**(3+4i); |

В результате переменной *Y* будет присвоено значение +5.

**Пример 3:**

*Модуль вещественного массива*

|  |  |
| --- | --- |
|  | **const** X = [-2, 2, -1, 3, -4];  Y = **abs**(x); |

В результате переменной *Y* будет присвоено значение массива [2 , 2 , 1 , 3 , 4].

### Функция выбора аргумента по номеру

**Синтаксис:**

*y* = **case**(*i, x0, x1, …, xn*);

**Аргументы:**

*i* – номер выбираемого аргумента, имеет тип integer,

*x0, x2, …, xn* – входные аргументы.

**Описание:**

*case(i, x0, x2, …, xn)* – функция выбора одного из входных аргументов *x0, x2, …, xn* по номеру, определяемому аргументом *i.* Нумерация аргументов начинается с нуля.

**Результат:**

*y* – переменная, содержащая выбранный входной аргумент.

**Пример:**

|  |  |
| --- | --- |
|  | **var**   i: **integer;**  **for** (i = 0, 4, 1) **begin**   y = **case**(i, 10, 20, 30, 40, 50);   // здесь y принимает значения от 10 до 50   // …  **end**; |

В результате переменной *y* будет присвоено последовательно значения от 10 до 50.

### Функция вычисления экспоненты вещественного или комплексного числа.

**Синтаксис:**

*y* **= exp***(x);*

**Аргументы:**

*x* – входное значение.

**Описание:**

*exp(x)* – функция вычисления экспоненты вещественного или комплексного числа. Входное значение может быть, как вещественным, так и комплексным числом. Входное значение может задаваться как заранее определенная переменная или как постоянное число. Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и комплексные части числа соответственно.

**Результат:**

*y* – экспонента входного значения *x*, определяется выражением e^*x*, где e – основание натурального логарифма.

**Пример 1:**

*Экспонента вещественного числа*

|  |  |
| --- | --- |
|  | **const** x = 5;  y = **exp**(x); |

В результате переменной *y* будет присвоено значение 148.41316.

**Пример 2:**

*Экспонента комплексного числа*

|  |  |
| --- | --- |
|  | y = **exp**(3+4i); |

В результате переменной *y* будет присвоено значение -13.128783-15.200784i.

### Функция вычисления десятичного логарифма вещественного или комплексного числа

**Синтаксис:**

*y* **= lg***(x);*

**Аргументы:**

*x* – входное значение.

**Описание:**

*lg(x)* – функция вычисления десятичного логарифма вещественного или комплексного числа. Входное значение может быть как вещественным, так и комплексным числом. Входное значение может задаваться как заранее определенная переменная или как постоянное число. Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и комплексные части числа соответственно.

**Результат:**

*y* – десятичный логарифм входного значения *x*.

**Пример 1:**

*Десятичный логарифм вещественного числа*

|  |  |
| --- | --- |
|  | **const** x = 5;    y = **lg**(x); |

В результате переменной *y* будет присвоено значение 0.69897.

**Пример 2:**

*Десятичный логарифм комплексного числа*

|  |  |
| --- | --- |
|  | y = **lg**(3+4i); |

В результате переменной *y* будет присвоено значение 0.69897+0.4027192i.

### Функция ограничения значения заданным интервалом

**Синтаксис:**

*xres* = **limit**(*x*,*xmin*,*xmax*);

**Аргументы:**

*x* – входное значение,

хmin – минимальное значение интервала,

*xmax* – максимальное значение интервала.

**Описание:**

*limit(x, xmin, xmax)* – функция ограничения значения *x* заданным интервалом *xmin, xmax*.

**Результат:**

*xres* – равен x при выполнении условий неравенства *xmin<x<xmax*

– равен *xmin* при выполнении условия x<= *xmin*

– равен *xmax* при выполнении условия x>= *xmax*

**Пример:**

|  |  |
| --- | --- |
|  | **input** x1;  **output** x2;    Min = 0.5;  Max = 0.7;    x2 = **limit**(x1, Min, Max); |

В результате выходная переменная блока x2 будет равна входной x1, но ограничена «снизу» и «сверху» значениями 0,5 и 0,7 включительно.

### Функция вычисления натурального логарифма вещественного или комплексного числа.

**Синтаксис:**

*y* **= ln***(x);*

**Аргументы:**

*x* – входное значение.

**Описание:**

*ln(x)* – функция вычисления натурального логарифма вещественного или комплексного числа. Входное значение может быть, как вещественным, так и комплексным числом. Входное значение может задаваться как заранее определенная переменная или как постоянное число. Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и комплексные части числа соответственно.

**Результат:**

*y* – натуральный логарифм входного значения *x*.

**Пример 1:**

*Натуральный логарифм вещественного числа*

|  |  |
| --- | --- |
|  | **const** x = 5;  y = **ln**(x); |

В результате переменной *y* будет присвоено значение 1.6094379.

**Пример 2:**

*Натуральный логарифм комплексного числа*

|  |  |
| --- | --- |
|  | y = **ln**(3+4i); |

В результате переменной *y* будет присвоено значение 1.6094379+0.92729522i.

### Функция вычисления максимального значения из двух значений или из значений вектора

**Синтаксис:**

*y* = **max***(a*, *b);*

*y* = **max***(X);*

**Аргументы:**

*a* – первое входное значение,

*b* – второе входное значение,

*Х* – входной массив, содержащий элементы вектора.

**Описание:**

*max(a, b)* – функция возвращает максимальное значение из *a* и *b*.

*max(X)* – функция возвращает максимальное значение из значений элементов входного вектора.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*y* = **max**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*y* = **max**([*x1,x2,x3,x4*]);

 как постоянный массив:

*y* = **max**([0,0,2,1,0]);

**Результат:**

*y* – максимальное значение.

**Пример 1:**

|  |  |
| --- | --- |
|  | **const** X = [2, 2, 1, 3, 4];  y = **max**(X); |

В результате переменной y будет присвоено значение 4, представляющее собой максимальное значение из элементов входного массива.

**Пример 2:**

|  |  |
| --- | --- |
|  | y = **max**(1, 5); |

В результате переменной y будет присвоено значение 5, представляющее собой максимальное значение из двух входных.

### Функция вычисления минимального значения из двух значений или из значений вектора

**Синтаксис:**

*y* = **min***(a*, *b);*

*y* = **min***(X);*

**Аргументы:**

*a* – первое входное значение,

*b* – второе входное значение,

*Х* – входной массив, содержащий элементы вектора.

**Описание:**

*min(a, b)* – функция возвращает минимальное значение из *a* и *b*.

*min(X)* – функция возвращает минимальное значение из значений элементов входного вектора.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*y* = **min**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*y* = **min**([*x1,x2,x3,x4*]);

 как постоянный массив:

*y* = **min**([0,0,2,1,0]);

**Результат:**

*y* – минимальное значение.

**Пример 1:**

|  |  |
| --- | --- |
|  | **const** X = [2, 2, 1, 3, 4];  y = **min**(X); |

В результате переменной y будет присвоено значение 1, представляющее собой минимальное значение из элементов входного массива.

**Пример 2:**

|  |  |
| --- | --- |
|  | y = **min**(1, 5); |

В результате переменной y будет присвоено значение 1, представляющее собой минимальное значение из двух входных.

### Функция вычисления значения линейной функции от заданного аргумента, заданной значениями функции и аргументов

**Синтаксис:**

*y* = **proportion***(x, x1*, *x2, y1, y2);*

**Аргументы:**

*x1, x2* – значения аргументов функции,

*y1, y2* – значения функции от аргументов *x1, x2*,

*х* – значение аргумента, от которого вычисляется значение линейной функции вида:

*y(x) = a0 + a1x*

**Описание:**

*proportion(x, x1, x2, y1, y2)* – функция возвращает значение функции от аргумента x, функция задана значениями *y1, y2* от аргументов *x1, x2*.

Функцию можно определить, решив уравнение



**Результат:**

*y* – значение функции от аргумента *x*.

**Пример :**

|  |  |
| --- | --- |
|  | **const** x = 10;  **const** x1 = 8;  **const** x2 = 6;  **const** y1 = 7;  **const** y2 = 9;  y = **proportion**(x, x1, x2, y1, y2); |

В результате переменной y будет присвоено значение 5, представляющее собой значение функции y = 15 – x при x = 10.

### Функция вычисления корня произвольной степени вещественного или комплексного числа

**Синтаксис:**

*z* **= root**(*y*, *x*)*;*

**Аргументы:**

*x* – входное значение,

*y* – степень корня.

**Описание:**

*root(x)* – функция вычисления корня произвольной степени вещественного или комплексного числа. Для вещественных чисел при чётных степенях входное значение должно быть положительным. Степень должна быть целым числом. Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и комплексные части числа соответственно.

**Результат:**

*z* – корень входного значения *x* степени *y*.

**Пример 1:**

*Корень произвольной степени вещественного числа*

|  |  |
| --- | --- |
|  | **const** y = 3;  **const** x = 5;  z = **root**(y, x); |

В результате переменной *z* будет присвоено значение 1.7099759.

**Пример 2:**

*Корень произвольной степени комплексного числа*

|  |  |
| --- | --- |
|  | **const** y = 3;  **const** x = 3+4i;  z = **root**(y, x); |

В результате переменной *z* будет присвоено значение 1.6289371+0.5201745i.

### Функция вычисления корня квадратного вещественного или комплексного числа

**Синтаксис:**

*y* **= sqrt***(x);*

**Аргументы:**

*x* – входное значение.

**Описание:**

*sqrt(x)* – функция вычисления корня квадратного вещественного или комплексного числа. Входное значение должно быть положительным числом. Входное значение может задаваться как заранее определенная переменная или как постоянное число. Входное значение может быть, как вещественным, так и комплексным числом. Если входное значение – вещественное число, то возвращаемое значение тоже вещественное число.

Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и комплексные части числа соответственно.

**Результат:**

*y* – корень квадратный от входного значения *x*.

**Пример 1:**

*Корень квадратный вещественного числа*

|  |  |
| --- | --- |
|  | **const** x = 5;  y = **sqrt**(x); |

В результате переменной *y* будет присвоено значение 2.236068.

**Пример 2:**

*Корень квадратный комплексного числа*

|  |  |
| --- | --- |
|  | y = **sqrt**(3+4i); |

В результате переменной *y* будет присвоено значение 2+1i.

## Логические операторы

### Оператор побитового логического И.

**Синтаксис:**

*c* **=** *a* **and** *b;*

**Аргументы:**

*a, b*– операнды побитового логического И.

**Описание:**

*and* – оператор побитового логического И. С входными операндами побитно выполняется операция логического И.

**Результат:**

*c* – результат операции.

**Пример:**

|  |  |
| --- | --- |
|  | y = 8; //1000  x = 5; //0101  z1 = y **and** x; //0000  y = 9; //1001  x = 5; //0101  z2 = y **and** x; //0001  y = 13; //1101  x = 5; //0101  z3 = y **and** x; //0101 |

В результате переменной *z1* будет присвоено значение 0, *z2* будет присвоено значение 1, *z3* будет присвоено значение 5.

### Оператор целочисленного деления.

**Синтаксис:**

*c* **=** *a* **div** *b;*

**Аргументы:**

*a*– делимое, целое число,

*b*– делитель, целое число.

**Описание:**

*div* – оператор целочисленного деления.

**Результат:**

*c* – результат операции, целое число.

**Пример:**

|  |  |
| --- | --- |
|  | y = 8;  x1 = 5;  x2 = 3;  x3 = 2;  z1 = y **div** x1;  z2 = y **div** x2;  z3 = y **div** x3; |

В результате переменной *z1* будет присвоено значение 1, так как если разделить целое число 8 на целое число 5, в результате получится целое число 1 и остаток 3. Переменной *z2* будет присвоено значение 2, так как если разделить целое число 8 на целое число 3, в результате получится целое число 2 и остаток 2. Переменной *z3* будет присвоено значение 4, так как если разделить целое число 8 на целое число 2, в результате получится целое число 4 и остаток 0.

### Оператор получения остатка от целочисленного деления.

**Синтаксис:**

*c* **=** *a* **mod** *b;*

**Аргументы:**

*a*– делимое, целое число,

*b*– делитель, целое число.

**Описание:**

*mod* – оператор получения остатка от целочисленного деления операндов.

**Результат:**

*c* – результат операции, целое число.

**Пример:**

|  |  |
| --- | --- |
|  | y = 8;  x1 = 5;  x2 = 3;  x3 = 2;  z1 = y **mod** x1;  z2 = y **mod** x2;  z3 = y **mod** x3; |

В результате переменной *z1* будет присвоено значение 3, так как если разделить целое число 8 на целое число 5, в результате получится целое число 1 и остаток 3. Переменной *z2* будет присвоено значение 2, так как если разделить целое число 8 на целое число 3, в результате получится целое число 2 и остаток 2. Переменной *z3* будет присвоено значение 0, так как если разделить целое число 8 на целое число 2, в результате получится целое число 4 и остаток 0.

### Оператор целочисленного или логического отрицания

**Синтаксис:**

*c* **= not** *expr;*

**Аргументы:**

*expr* – выражение для логического отрицания или число для целочисленного отрицания.

**Описание:**

*not* – оператор целочисленного или логического отрицания. При операции логического отрицания выражение *expr* может содержать любые математические или логические операции. Результат математической или логической операции будет логическая переменная, имеющая значение «ДА» или «НЕТ». После операции логического отрицания переменная, имеющая значение «ДА», сменит его на «НЕТ», и наоборот. Результирующая переменная должна иметь тип *boolean*. Применение операции отрицания к данным целочисленных типов вызывает побитную инверсию (отрицание) соответствующего данному числу двоичного кода.

**Результат:**

*c* – результат операции.

**Пример:**

|  |  |
| --- | --- |
|  | **var**  c2:**boolean**,  c3:**boolean**  **end**  x = 78;  c1 = **not** x;  c2 = **not** (x< 10);  c3 = **not** (x > 10); |

В результате переменной *c1* будет присвоено значение -79. Число 78 занимает в памяти 2 байта и в двоичной форме имеет вид: 0000000001001110. При выполнении операции все 0 заменяются на 1, а 1 - на 0 соответственно. После выполнения операции not 78 получим: 1111111110110001, что соответствует числу -79 в десятичной форме. Переменной *c2* будет присвоено значение «НЕТ», переменной *c3* будет присвоено значение «ДА».

### Оператор побитового логического ИЛИ

**Синтаксис:**

*c* **=** *a* **or** *b;*

**Аргументы:**

*a, b*– операнды побитового логического ИЛИ.

**Описание:**

*or* – оператор побитового логического ИЛИ. С входными операндами побитно выполняется операция логического ИЛИ.

**Результат:**

*c* – результат операции.

**Пример:**

|  |  |
| --- | --- |
|  | y = 8; //1000  x = 5; //0101  z1 = y **or** x; //1101  y = 9; //1001  x = 5; //0101  z2 = y **or** x; //1101  y = 13; //1110  x = 5; //0101  z3 = y **or** x; //1111 |

В результате переменной *z1* будет присвоено значение 13, *z2* будет присвоено значение 13, *z3* будет присвоено значение 16.

### Оператор побитового сдвига влево.

**Синтаксис:**

*c* **=***a* **shl** *b;*

**Аргументы:**

*a* – операнд побитового сдвига влево,

*b* – количество сдвигов.

**Описание:**

*shl* – оператор побитового сдвига влево. Все единичные биты операнда сдвигаются на заданное количество битов влево, 31-й бит определяет знак числа.

**Результат:**

*c* – результат операции.

**Пример 1:**

|  |  |
| --- | --- |
|  | a = 1;//0001  b = 3;  c = a **shl** b; //1000 |

В результате переменной *c* будет присвоено значение 8.

### Оператор побитового сдвига вправо.

**Синтаксис:**

*c* **=** *a* **shr** *b;*

**Аргументы:**

*a* – операнд побитового сдвига вправо,

*b* – количество сдвигов.

**Описание:**

*shl* – оператор побитового сдвига вправо. Все единичные биты операнда сдвигаются на заданное количество битов вправо, 31-й бит определяет знак числа.

**Результат:**

*c* – результат операции.

**Пример 1:**

|  |  |
| --- | --- |
|  | a = 8;//1000  b = 3;  c = a **shr** b;//0001 |

В результате переменной *z* будет присвоено значение 1.

### Оператор побитового логического ИСКЛЮЧАЮЩЕГО ИЛИ

**Синтаксис:**

*c* **=** *a* **xor** *b;*

**Аргументы:**

*a, b*– операнды побитового логического ИСКЛЮЧАЮЩЕГО ИЛИ.

**Описание:**

*or* – оператор побитового логического ИСКЛЮЧАЮЩЕГО ИЛИ. С входными операндами побитно выполняется операция логического ИСКЛЮЧАЮЩЕГО ИЛИ.

**Результат:**

*c* – результат операции.

**Пример:**

|  |  |
| --- | --- |
|  | y = 8; //1000  x = 5; //0101  z1 = y **xor** x; //1101  y = 9; //1001  x = 5; //0101  z2 = y **xor** x; //1100  y = 13; //1101  x = 5; //0101  z3 = y **and** x; //1000 |

В результате переменной *z1* будет присвоено значение 13, *z2* будет присвоено значение 12, *z3* будет присвоено значение 8.

## Векторные и матричные

### Первый ненулевого элемент в массиве

**Синтаксис:**

*x =* **activeelement***(X);*

**Аргументы:**

*Х* – входной массив

**Описание:**

*activeelement(X)* – функция возвращает индекс первого ненулевого элемента в массиве. Если все элементы нулевые, то возвращает ноль. Индексы начинаются с единицы. Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*х* = **activeelement**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*x* = **activeelement**([*x1,x2,x3,x4*]);

 как постоянный массив:

*х* = **activeelement**([0,0,2,1,0]);

**Результат:**

*x* – индекс первого ненулевого элемента в массиве. Имеет тип integer.

**Пример 1:**

|  |  |
| --- | --- |
|  | **output** i;//выход – индекс ненулевого элемента  i =**activeelement**([0,1,0]); // Результат: i = 2 |

В результате переменной i будет присвоено целое число 2, соответствующее индексу первого ненулевого элемента в массиве [0,1,0].

**Пример 2:**

|  |  |
| --- | --- |
|  | **output** i;//выход – индекс ненулевого элемента  i =**activeelement**([0,0,2,1,0]); // Результат: i = 3 |

В результате переменной i будет присвоено целое число 3, соответствующее индексу первого ненулевого элемента в массиве [0,0,2,1,0].

**Пример 3:**

|  |  |
| --- | --- |
|  | **output** i;//выход – индекс ненулевого элемента  **const** a = [0,0,0,0,0,0];  i =**activeelement**(a); // Результат: i = 0 |

В результате переменной i будет присвоено целое число 0, так как все элементы в массиве [0,0,0,0,0,0] нулевые.

### Количество столбцов матрицы или количество элементов массива

**Синтаксис:**

*x* = **cols**(*X*);

*c* = **cols**(*M*);

**Аргументы:**

*Х* – входной массив.

*M* – входная матрица.

**Описание:**

*cols(X)* – функция возвращает количество элементов массива.

*cols(M)* – функция возвращает количество столбцов матрицы.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*x* = **cols**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*x* = **cols***(*[*x1,x2,x3,x4*]);

 как постоянный массив:

*x* = **cols**([-1.80, -1.60, -1.40, -1.20]);

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*c* = **cols**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*c* = **cols**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянный массив:

*c* = **cols**([[1,2],[3,4],[5,6]]);

**Результат:**

*x* – количество элементов массива *Х.* Имеет тип *integer*,

*c* – количество столбцов матрицы *M.* Имеет тип *integer.*

**Пример 1:**

|  |  |
| --- | --- |
|  | **output** na; //выход – количество элементов массива  a = [1,2,3,4,5];  na = **cols**(a); // na = 5 |

В результате переменной *na* будет присвоено целое число 5, соответствующее количеству элементов массива *a*.

**Пример 2:**

|  |  |
| --- | --- |
|  | **output** nb; //выход – количество столбцов матрицы  b = [[1,2],[3,4],[5,6]]; //матрица 2x3  nb = **cols**(b); //nb = 2 |

В результате переменной *nb* будет присвоено целое число 2, соответствующее количеству столбцов матрицы *b*.

### Умножение полиномов

**Синтаксис:**

*С* = **conv**(*A, B*);

**Аргументы:**

*A, B* – входные массивы, содержащие коэффициенты a0, a1, …an, b0, b1, …bn полиномов вида:

f(t) = a0+a1t+a2t2+…+antn

g(t) = b0+b1t+b2t2+…+bntn

**Описание:**

*conv(A, B)* – функция возвращает массив коэффициентов полинома, полученного в результате умножения полинома, заданного массивом коэффициентов *А* на полином, заданный массивом коэффициентов *B*.

Входные массивы *A, B* могут задаваться:

 как переменные типа массив, определенные ранее:

*С* = **conv**(*A,B*);

 как массивы, состоящие из переменных, определенных ранее:

*С* = **conv**([*a1,a2,a3,a4*],[*b1,b2,b3,b4*]);

 как постоянные массивы:

*С* = **conv**([-1, -6, -4, -2],[-4, 7, 5, -3]);

**Результат:**

*С* – выходной массив, содержащий коэффициенты полинома, полученного в результате умножения полинома, заданного массивом коэффициентов *А* на полином, заданный массивом коэффициентов *B*:

c(t) = f(t)g(t) = c0+c1t+c2t2+…+c2nt2n

**Пример:**

|  |  |
| --- | --- |
|  | **const** A = [2, 3, 5];  **const** B = [-1, -2, 2];  C = **conv**(A, B); //[-2, -7, -7, -4, 10] |

В результате элементам массива C будут присвоены значения [-2, -7, -7, -4, 10], являющиеся коэффициентами полинома c(t) = -2-7t-7t2-4t3+10t4, полученного в результате умножения полинома f(t) = 2+3t+5t2 на полином g(t) = -1-2t+2t2. Полиномы f(t) и g(t) определенны массивами *A* и *B*.

### Деление полиномов

**Синтаксис:**

*С* = **deconv**(*A, B, D*);

**Аргументы:**

*A, B* – входные массивы, содержащие коэффициенты a0, a1, …a2n, b0, b1, …bn полиномов вида:

f(t) = a0+a1t+a2t2+…+a2nt2n

g(t) = b0+b1t+b2t2+…+bntn

*D* – выходной массив, содержащий коэффициенты полинома остатка от деления.

**Описание:**

*deconv(A, B, D)* – функция возвращает массив коэффициентов полинома, полученного в результате деления полинома, заданного массивом коэффициентов *А* на полином, заданный массивом коэффициентов *B*. Массив коэффициентов полинома остатка от деления возвращается в переменную *D*.

Входные массивы *A, B* могут задаваться:

 как переменные типа массив, определенные ранее:

*С* = **deconv**(*A, B, D*);

 как массивы, состоящие из переменных, определенных ранее:

*С* = **deconv***(*[*a1,a2,a3,a4*],[*b1,b2,b3,b4*], *D*);

 как постоянные массивы:

*С* = **deconv**([-1, -6, -4, -2],[-4, 7, 5, -3], *D*);

**Результат:**

*С* – выходной массив, содержащий коэффициенты полинома, полученного в результате деления полинома, заданного массивом коэффициентов *А* на полином, заданный массивом коэффициентов *B*:

q(t) = f(t)/g(t) = c0+c1t+c2t2+…+cntn

**Пример 1:**

|  |  |
| --- | --- |
|  | **var** D:array;  **const** A = [1, 2, -1,0, 1, 2];  **const** B = [-1, -1, 2, 1];  C = **deconv**(A, B, D); |

В результате элементам массива *C* будут присвоены значения [8, -3, 1], являющиеся коэффициентами полинома q(t) = 8 - 3t + 2t2, полученного в результате деления полинома f(t) = 1 + 2t - t2 + t4 + 2t5 на полином g(t) = -1 – t + 2t2 + t3. Полиномы f(t) и g(t) определенны массивами *A* и *B*. Элементам массива *D* будут присвоены значения [9, 7, -18], являющиеся коэффициентами полинома остатка от деления r(t) = 9 + 7t - 18t2.

### Вычисление детерминанта матрицы

**Синтаксис:**

*d* = **det**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*det(M)* – функция возвращает значение детерминанта матрицы. Матрица *M* должна быть квадратной.

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*d* = **det**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*d* = **det**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянный массив:

*d* = **det**([[1,2],[3,4],[5,6]]);

**Результат:**

*d* – значение детерминанта матрицы *M.*

**Пример :**

|  |  |
| --- | --- |
|  | **const** M = [[2, 2, 1, 3, 4], [3, 1, 2, 3, 1], [4, -1, 2, 4, -2], [1, -1, 1, 1, 2], [4, -1, 2, 5, 6]];  d = **det**(M); //15 |

В результате переменной d будет присвоено значение детерминанта матрицы *М*, равное 15.

### Создание диагональной матрицы

**Синтаксис:**

*M* = **diag**(*X*);

**Аргументы:**

*Х* – входной массив.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*М* = **diag**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*M* = **diag***(*[*x1,x2,x3,x4*]);

 как постоянный массив:

*M* = **diag**([-1.80, -1.60, -1.40, -1.20]);

**Описание:**

*diag(X)* – функция возвращает квадратную матрицу, у которой диагональные элементы равны элементам вектора *X* (диагональную матрицу)*.* Все остальные элементы матрицы, стоящие вне главной диагонали, равны нулю.

**Результат:**

*М* – диагональная матрица*.* Матрица *М* -квадратная.

**Пример:**

|  |  |
| --- | --- |
|  | **const** X = [2, 2, 1, 3, 4];  M = **diag**(X); |

В результате переменной *M* будет присвоено значение массива

[

[2, 0, 0, 0, 0],

[0, 2, 0, 0, 0],

[0, 0, 1, 0, 0],

[0, 0, 0, 3, 0],

[0, 0, 0, 0, 4]

],

определяющего диагональную матрицу, у которой диагональные элементы равны элементам вектора *X*.

### Вычисление собственных чисел вещественной матрицы

**Синтаксис:**

*X* = **eig**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*eig(M)* – функция возвращает массив собственных чисел матрицы. Матрица *M* должна быть квадратной и иметь вещественные элементы. Вектор собственных чисел матрицы порядка n содержит n чисел, часть из которых могут быть одинаковыми. Соответствует выражению *polyroots(poly(M))*.

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*X* = **eig**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*X* = **eig**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянная матрица:

*X* = **eig**([[1,2],[3,4],[5,6]]);

**Результат:**

*X* – массив вещественных собственных чисел матрицы *M.*

**Пример:**

|  |  |
| --- | --- |
|  | **сonst** M = [[1, -3, 4], [4, -7, 8], [6, -7, 7]];  X = **eig**(M); //[-1, -1, 3] |

В результате элементам массива *X* будут присвоены значения [-1, -1, 3], являющиеся вещественными собственными числами матрицы *M*.

### Извлечение элементов из матрицы или вектора по номерам

**Синтаксис:**

*C* = **extract**(*M, A, B*);

*D* = **extract**(*X, Y*);

**Аргументы:**

*M* – входная матрица,

*A* – входной массив номеров строк извлекаемых элементов матрицы *M*,

*B* – входной массив номеров столбцов извлекаемых элементов матрицы *M*,

*X* – входной вектор,

*Y* – входной массив номеров извлекаемых элементов вектора *X*.

**Описание:**

*extract(M, A, B)* – функция извлечения из матрицы элементов, определяемых номерами строк и столбцов.

*extract(X, Y)* – функция извлечения из вектора элементов, определяемых номерами в векторе.

Входные массивы *X*, *Y*, *A*, *B* могут задаваться:

 как переменные типа массив, определенные ранее:

*D* = **extract**(*X,Y*);

 как переменные типа массив, состоящие из переменных *double*, определенных ранее:

*D* = **extract***(*[*x1,x2,x3,x4*]*,*[*y1,y2,y3,y4*]);

*D* = **extract**([*x1,x2,x3,x4*],Y);

*D* = **extract** (X,[*y1,y2,y3,y4*]);

 как постоянные массивы:

*D* = **extract** ([-1.80, -1.60, -1.40, -1.20],*Y*);

*D* = **extract** (*X*,[-1.80, -1.60, -1.40, -1.20]);

*D* = **extract** ([-1.80, -1.60, -1.40, -1.20],[-1.40, -0.78,-0.53,-0.35]);

**Результат:**

*С* – выходная матрица, содержащая элементы входной матрицы, полученные в результате извлечения из входной матрицы элементов, определяемых номерами строк и столбцов,

*D* – выходной массив, содержащий элементы входного вектора, определяемые номерами элементов.

**Пример 1:**

*Извлечение элементов из матрицы:*

|  |  |
| --- | --- |
|  | //массивы элементов матрицы  **const** M = [[1, -3, 4], [4, -7, 8], [6, -7, 7]];  C = **extract**(M, [2, 3], [1, 3]); |

В результате элементам матрицы *C* будут присвоены значения [[4 , 8] [6, 7]].

**Пример 2**

*Извлечение элементов из вектора:*

|  |  |
| --- | --- |
|  | //массивы элементов вектора  **const** X = [1, -3, 4, 4, -7, 8, 6, -7, 7];  D = **extract**(X, [2, 3]); |

В результате элементам массива *D* будут присвоены значения [-3, 4], соответствующие элементам массива *X* с номерами 2 и 3.

### Создание единичной матрицы

**Синтаксис:**

*M* = **eye**(*n*);

**Аргументы:**

*n* – размерность единичной матрицы.

**Описание:**

*eye(n)* – функция возвращает матрицу размерностью n на n, у которой все диагональные элементы равны единице.

**Результат:**

*М* – единичная матрица*.* Матрица *М* –квадратная.

**Пример:**

|  |  |
| --- | --- |
|  | **const** n = 5;  M = **eye**(n); |

В результате переменной *M* будет присвоено значение массива

[

[1, 0, 0, 0, 0],

[0, 1, 0, 0, 0],

[0, 0, 1, 0, 0],

[0, 0, 0, 1, 0],

[0, 0, 0, 0, 1]

],

определяющего единичную матрицу, у которой все диагональные элементы равны 1.

### Вычисление обратной матрицы

**Синтаксис:**

*C* = **inv**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*inv(M)* – функция возвращает матрицу, являющуюся обратной матрице *M*. Матрица *M* должна быть квадратной. При умножении обратной матрицы на исходную матрицу получается единичная матрица (все диагональные элементы которой равны 1). Для вычисления обратной матрицы можно использовать операцию деления матриц.

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*C* = **inv**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*C* = **inv**([[*x1,x2*],[*x3,x4*]]);

 как постоянная матрица:

*C* = **inv**([[1,2],[3,4]]);

**Результат:**

*С* – выходная матрица, являющаяся обратной матрице *M*.

**Пример:**

|  |  |
| --- | --- |
|  | //массивы элементов матрицы  **const** M = [[2, 5, 7], [6, 3, 4], [5, -2, -3]];  C = **inv**(M); |

В результате элементам матрицы *C* будут присвоены значения [[1, -1, 1], [-38, 41, -34], [27, -29, 24]], определяющие обратную для матрицы *M* матрицу *C*.

### Сортировка вектора по убыванию значений

**Синтаксис:**

*Y* = **invsort***(X);*

**Аргументы:**

*Х* – входной массив, содержащий элементы вектора.

**Описание:**

*invsort(X)* – производится сортировка массива по убыванию элементов. Для комплексных векторов сортировка ведётся по модулям чисел.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*Y* = **invsort**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*Y* = **invsort**([*x1,x2,x3,x4*]);

 как постоянный массив:

*Y* = **invsort**([0,0,2,1,0]);

**Результат:**

*Y* – выходной массив, содержащий элементы отсортированного вектора.

**Пример:**

|  |  |
| --- | --- |
|  | **сonst** X = [2, 2, 1, 3, 4];  Y = **invsort**(X); |

В результате элементам массива *Y* будут присвоены значения [4, 3, 2, 2, 1], представляющие собой отсортированные по убыванию элементы входного массива.

### Сортировка векторов по убыванию значений определяющего вектора

**Синтаксис:**

**invsortby***(X, Y1, Y2,..,Yn);*

**Аргументы:**

*Х* – входной массив, содержащий элементы сортируемого вектора определяющего сортировку вектора,

*Y1,..,Yn* – входные массивы, содержащие элементы сортируемых векторов.

**Описание:**

*invsortby(X, Y1, Y2,..,Yn)* – производится сортировка множества векторов по убыванию значений первого из них (т.е. первый указанный вектор является определяющим, а элементы остальных переставляются в соответствии с порядком перестановки элементов первого). Для комплексных векторов сортировка ведётся по модулям чисел.

Входные массивы *X, Y1, Y2,..,Yn* могут задаваться:

 как переменные типа массив, определенные ранее:

**invsortby**(*X, Y1, Y2,..,Yn*);

 как массивы, состоящие из переменных, определенных ранее:

**invsortby**([*x1,x2,x3,x4*], *Y1, Y2,..,Yn*);

 как постоянные массивы:

**invsortby**([0,0,2,1,0], *Y1, Y2,..,Yn*);

**Пример 1:**

|  |  |
| --- | --- |
|  | X = [2, 3, 1];  Y = [7, 0, 4];  **invsortby**(X, Y); |

В результате элементам массива *Х* будут присвоены значения [3, 2, 1], представляющие собой отсортированные по убыванию элементы исходного массива *X*, элементам массива *Y* будут присвоены значения [0, 7, 4], представляющие собой отсортированные в соответствии с сортировкой определяющего массива элементы исходного массива *Y*.

**Пример 2:**

|  |  |
| --- | --- |
|  | X = [2, 3, 1];  Y1 = [7, 0, 4];  Y2 = [5, 2, 7];  Y3 = [-1, 0, 8];  **invsortby**(X, Y1, Y2, Y3); |

В результате элементам массива *Х* будут присвоены значения [3, 2, 1], представляющие собой отсортированные по убыванию элементы исходного массива *X*, элементам массивов *Y1, Y2, Y3* будут присвоены значения [0, 7, 4], [2, 5, 7], [0, -1, 8] соответственно, представляющие собой отсортированные в соответствии с сортировкой определяющего массива элементы исходных массивов *Y1, Y2, Y3*.

### Вычисление вектора арифметической прогрессии

**Синтаксис:**

**Y = linspace***(xmin* ,*xmax*, *n);*

**Аргументы:**

хmin – минимальное значение арифметической прогрессии,

*xmax* – максимальное значение арифметической прогрессии,

*n* – размерность вектора арифметической прогрессии.

**Описание:**

*linspace(xmin ,xmax, n)* – функция вычисления вектора арифметической прогрессии от значения *xmin* до значения *xmax*. Размерность вектора определяется параметром *n*.

**Результат:**

*Y* – выходной массив, содержащий значения вектора арифметической прогрессии.

**Пример:**

|  |  |
| --- | --- |
|  | **const** n=81;  **output** x[n];  minimum = 20.0;  maximum = 260.0;  x = **linspace**(minimum, maximum, n); |

В результате элементам массива *Х* будут присвоены значения [20, 23, 26,…, 257, 260], представляющие собой значения арифметической прогрессии, размерность массива – 81 элемент.

Точное значение полученного массива *Х* = [20 , 23 , 26 , 29 , 32 , 35 , 38 , 41 , 44 , 47 , 50 , 53 , 56 , 59 , 62 , 65 , 68 , 71 , 74 , 77 , 80 , 83 , 86 , 89 , 92 , 95 , 98 , 101 , 104 , 107 , 110 , 113 , 116 , 119 , 122 , 125 , 128 , 131 , 134 , 137 , 140 , 143 , 146 , 149 , 152 , 155 , 158 , 161 , 164 , 167 , 170 , 173 , 176 , 179 , 182 , 185 , 188 , 191 , 194 , 197 , 200 , 203 , 206 , 209 , 212 , 215 , 218 , 221 , 224 , 227 , 230 , 233 , 236 , 239 , 242 , 245 , 248 , 251 , 254 , 257 , 260].

### Вычисление вектора арифметической прогрессии в логарифмической шкале

**Синтаксис:**

Y = **logspace***(xmin* ,*xmax*, *n);*

**Аргументы:**

хmin – минимальне значение арифметической прогрессии,

*xmax* – максимальное значение арифметической прогрессии,

*n* – размерность вектора арифметической прогрессии.

**Описание:**

*logspace(xmin ,xmax, n)* – функция вычисления вектора арифметической прогрессии в логарифмической шкале от значения *xmin* до значения *xmax*. Количество элементов арифметической прогрессии определяется параметром *n*. В результате формируется вектор, содержащий элементы со значениями от 10^ *xmin* до 10^ *xmax*. Вектор арифметической прогрессии в линейной шкале от значения *xmin* до значения *xmax* размером *n* вычисляется при помощи ф-ии *linspace(xmin ,xmax, n)*.

**Результат:**

*Y* – выходной массив, содержащий элементы вектора со значениями от 10^ *xmin* до 10^ *xmax*, где числа от *xmin* до *xmax* – значения арифметической прогрессии.

**Пример:**

|  |  |
| --- | --- |
|  | **const** n=81;  **output** x[n];  **output** y[n];  minimum = 20;  maximum = 260;  y = **linspace**(minimum, maximum, n);  x = **logspace**(minimum, maximum, n); |

В результате элементам массива *y* будут присвоены значения [20, 23, 26,…, 257, 260], представляющие собой значения арифметической прогрессии, размерность массива – 81 элемент. Элементам массива *x* будут присвоены значения [1e20, 1e23, 1e26,…, le257, 1e260], где степень представляет собой значения арифметической прогрессии, размерность массива – 81 элемент.

Точное значение полученного массива *y* = [20 , 23 , 26 , 29 , 32 , 35 , 38 , 41 , 44 , 47 , 50 , 53 , 56 , 59 , 62 , 65 , 68 , 71 , 74 , 77 , 80 , 83 , 86 , 89 , 92 , 95 , 98 , 101 , 104 , 107 , 110 , 113 , 116 , 119 , 122 , 125 , 128 , 131 , 134 , 137 , 140 , 143 , 146 , 149 , 152 , 155 , 158 , 161 , 164 , 167 , 170 , 173 , 176 , 179 , 182 , 185 , 188 , 191 , 194 , 197 , 200 , 203 , 206 , 209 , 212 , 215 , 218 , 221 , 224 , 227 , 230 , 233 , 236 , 239 , 242 , 245 , 248 , 251 , 254 , 257 , 260].

Точное значение полученного массива*x* = [1E20 , 1E23 , 1E26 , 1E29 , 1E32 , 1E35 , 1E38 , 1E41 , 1E44 , 1E47 , 1E50 , 1E53 , 1E56 , 1E59 , 1E62 , 1E65 , 1E68 , 1E71 , 1E74 , 1E77 , 1E80 , 1E83 , 1E86 , 1E89 , 1E92 , 1E95 , 1E98 , 1E101 , 1E104 , 1E107 , 1E110 , 1E113 , 1E116 , 1E119 , 1E122 , 1E125 , 1E128 , 1E131 , 1E134 , 1E137 , 1E140 , 1E143 , 1E146 , 1E149 , 1E152 , 1E155 , 1E158 , 1E161 , 1E164 , 1E167 , 1E170 , 1E173 , 1E176 , 1E179 , 1E182 , 1E185 , 1E188 , 1E191 , 1E194 , 1E197 , 1E200 , 1E203 , 1E206 , 1E209 , 1E212 , 1E215 , 1E218 , 1E221 , 1E224 , 1E227 , 1E230 , 1E233 , 1E236 , 1E239 , 1E242 , 1E245 , 1E248 , 1E251 , 1E254 , 1E257 , 1E260].

### Решение системы линейных алгебраических уравнений (СЛАУ)

**Синтаксис:**

*Z* = **lsolve**(*X,Y*);

**Аргументы:**

*X* – входной массив элементов [[a11, ..,a1n],…,[ am1, ..,amn]] , определяющий матрицу системы линейных алгебраических уравнений А,

*Y* – входной массив элементов [b11, ..,b1m], определяющий вектор B системы линейных алгебраических уравнений А.

A :

**Описание:**

*lsolve(X, Y)* – функция решения системы линейных алгебраических уравнений (СЛАУ).

Входные массивы *X*, *Y* могут задаваться:

 как переменные типа массив, определенные ранее:

*Z* = **lsolve**(*X, Y*);

 как переменные типа массив, состоящие из переменных *double*, определенных ранее:

*Z* = **lsolve**([[*x11,x12,x13,x14*],[*x21,x22,x23,x24*], [*x31,x32,x33,x34*], [*x41,x42,x43,x44*]]*,*[*y1,y2,y3,y4*]);

*Z* = **lsolve**(([[*x11,x12,x13,x14*],[*x21,x22,x23,x24*], [*x31,x32,x33,x34*], [*x41,x42,x43,x44*]],*Y*);

*Z* = **lsolve**(*X*,[*y1,y2,y3,y4*]);

 как постоянные массивы:

*Z* = **lsolve**([[1, 2],[2, -3]],*Y*);

*Z* = **lsolve**(*X*,[3, -1]);

*Z* = **lsolve**([[1, 2],[2, -3]], [3,-1]);

**Результат:**

*Z* – выходной массив элементов [x1, ..,xn], определяющий вектор решений системы линейных алгебраических уравнений А.

**Пример:**

|  |  |
| --- | --- |
|  | //массивы элементов матрицы  **const** X = [[1, 2], [2, -3]];  Y = **lsolve**(X, [3, -1]); |

В результате элементам массива *Y* будут присвоены значения [1, 1], являющиеся решениями системы уравнений:



### Cоздание нулевой матрицы

**Синтаксис:**

*M* = **matrix**(*n, m*);

**Аргументы:**

*n* – количество строк нулевой матрицы,

*m* – количество столбцов нулевой матрицы.

**Описание:**

*matrix(n, m)* – функция возвращает матрицу размерностью n на m, у которой все элементы равны нулю.

**Результат:**

*М* – возвращаемая нулевая матрица*.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** n = 5;  M = **matrix**(n, 4); |

В результате переменной *M* будет присвоено значение массива

[

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]

],

определяющего нулевую матрицу, у которой все элементы равны 0.

### Создание матрицы единиц

**Синтаксис:**

*M* = **matrix1**(*n, m*);

**Аргументы:**

*n* – количество строк матрицы единиц,

*m* – количество столбцов матрицы единиц.

**Описание:**

*matrix1(n, m)* – функция возвращает матрицу размерностью n на m, у которой все элементы равны единице.

**Результат:**

*М* – возвращаемая матрица единиц*.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** n = 5;  M = **matrix1**(n, 4); |

В результате переменной *M* будет присвоено значение массива

[

[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]

],

определяющего матрицу единиц, у которой все элементы равны 1.

### Размножение каждого элемента вектора

**Синтаксис:**

*Y* **= multiply***(n, X);*

**Аргументы:**

*n* – множитель для элементов вектора ,

*Х* – входной массив, содержащий элементы вектора.

**Описание:**

*multiply(n, X)* – функция размножения элементов входного вектора в *n* раз.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

Y = **multiply**(n, *X*);

 как массив, состоящий из переменных, определенных ранее:

Y = **multiply**(n, [*x1,x2,x3,x4*]);

 как постоянный массив:

Y = **multiply**(n, [1,2,3]);

**Пример:**

|  |  |
| --- | --- |
|  | **const** X = [1, 2, 3];  **Y = multiply**(2, X); |

В результате элементам массива *Х* будут присвоены значения [1, 1, 2, 2, 3, 3], представляющие собой размноженные в 2 раза элементы входного массива.

### Вычисление первой нормы вещественной матрицы

**Синтаксис:**

*x* = **norm1**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*norm1(M)* – функция возвращает первую норму вещественной матрицы. Суммируются все элементы в столбцах матрицы *M*, взятые по модулю, и максимальная из полученных сумм объявляется нормой (первая норма).

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*x* = **norm1**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*x* = **norm1**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянная матрица:

*x* = **norm1**([[1,2],[3,4],[5,6]]);

**Результат:**

*х* – значение первой нормы матрицы *M.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** M = [  [1, -3, 2],  [4, 5, -1],  [3, 8, -6]  ];  x = **norm1**(M); //16 |

В результате переменной *x* будет присвоено значение первой нормы матрицы *М*, равное 16.

### Вычисление бесконечной нормы вещественной матрицы

**Синтаксис:**

*x* = **normi**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*normi(M)* – функция возвращает бесконечную норму вещественной матрицы. Суммируются, по модулю, все строки матрицы *M* и максимальная из полученных сумм объявляется нормой (бесконечная норма).

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*x* = **normi**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*x* = **normi**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянный массив:

*x* = **normi**([[1,2],[3,4],[5,6]]);

**Результат:**

*х* – значение бесконечной нормы матрицы *M.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** M = [[1, -3, 2], [4, 5, -1], [3, 8, -6];  x = **normi**(M); //17 |

В результате переменной *x* будет присвоено значение бесконечной нормы матрицы *М*, равное 17.

### Вычисление характеристического полинома вещественной матрицы

**Синтаксис:**

*X* = **poly**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*poly(M)* – функция возвращает массив коэффициентов характеристического полинома матрицы *М*. Матрица *M* должна быть квадратной и иметь вещественные элементы.

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*X* = **poly**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*X* = **poly**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянный массив:

*X* = **poly**([[1,2],[3,4],[5,6]]);

**Результат:**

*X* – выходной массив, содержащий коэффициенты характеристического полинома матрицы *М*:

 a0 + a1x + a2x2 +…+ anxn

**Пример:**

|  |  |
| --- | --- |
|  | **const** M = [[1, -3, 4], [4, -7, 8], [6, -7, 7]];  X = **poly**(M); //[3, 5, 1, -1] |

В результате элементам массива *X* будут присвоены значения [3, 5, 1, -1], являющиеся корнями характеристического полинома 3 + 5x + x2 - x3 вещественной матрицы *M*.

### Сложение полиномов

**Синтаксис:**

*С* = **polyadd**(*A, B*);

**Аргументы:**

*A,B* – входные массивы, содержащие коэффициенты a0, a1, …an, b0, b1, …bn полиномов вида:

f(t) = a0+a1t+a2t2+…+antn

g(t) = b0+b1t+b2t2+…+bntn

**Описание:**

*polyadd(A, B)* – функция возвращает массив коэффициентов полинома, полученного в результате сложения полинома, заданного массивом коэффициентов *А* с полиномом, заданным массивом коэффициентов *B*.

Входные массивы *A, B* могут задаваться:

 как переменные типа массив, определенные ранее:

*С* = **polyadd**(*A, B*);

 как массивы, состоящие из переменных, определенных ранее:

*С* = **polyadd**([*a1,a2,a3,a4*],[*b1,b2,b3,b4*]);

 как постоянные массивы:

*С* = **polyadd**([-1, -6, -4, -2],[-4, 7, 5, -3]);

**Результат:**

*С* – выходной массив, содержащий коэффициенты полинома, полученного в результате сложения полинома, заданного массивом коэффициентов *А* с полиномом, заданным массивом коэффициентов *B*:

q(t) = f(t)+g(t) = c0+c1t+c2t2+…+cntn

**Пример 1:**

|  |  |
| --- | --- |
|  | **const** A = [1, 2, -1, 0, 1, 2];  **const** B = [-1, -1, 2, 1];  C = **polyadd**(A, B); |

В результате элементам массива *C* будут присвоены значения [0, 1, 1, 1, 1, 2], являющиеся коэффициентами полинома q(t) = t+t2+ t3 + t4+2t5, полученного в результате сложения полинома f(t) = 1+2t-t2+t4+2t5 с полиномом g(t) = -1-t+2t2+t3. Полиномы f(t) и g(t) определенны массивами *A* и *B*.

### Вычисление производной от полинома

**Синтаксис:**

*С* = **polyder**(*A*);

**Аргументы:**

*A* – входной массив, содержащий коэффициенты a0, a1, …an полинома вида:

f(t) = a0+a1t+a2t2+…+antn

**Описание:**

*polyder(A)* – функция возвращает массив коэффициентов полинома, полученного в результате вычисления производной от полинома, заданного массивом коэффициентов *А*.

Входной массив *A* может задаваться:

 как переменная типа массив, определенная ранее:

*С* = **polyder**(*A*);

 как массив, состоящий из переменных, определенных ранее:

*С* = **polyder**([*a1,a2,a3,a4*]);

 как постоянный массив:

*С* = **polyder**([-1, -6, -4, -2]);

**Результат:**

*С* – выходной массив, содержащий коэффициенты c0, c1, …cn-1 полинома, полученного в результате вычисления производной от полинома, заданного массивом коэффициентов *А*:

q(t) = f’(t) = c0+c1t+c2t2+…+cn-1tn-1

**Пример:**

|  |  |
| --- | --- |
|  | **const** A = [1, 0, -2, -5];  C = **polyder**(A); |

В результате элементам массива *C* будут присвоены значения [0, -4, -15], являющиеся коэффициентами полинома q(t) = 0 - 4t - 15t2, полученного в результате вычисления производной от полинома f(t) = 1 - 2t2 - 5t3. Полином f(t) определен массивом *A*.

### Вычисление корней характеристического полинома вещественной матрицы

**Синтаксис:**

*X* = **polyroots**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*polyroots(M)* – функция возвращает массив корней характеристического полинома, являющихся собственными числами матрицы *М*. Матрица *M* должна быть квадратной и иметь вещественные элементы.

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*X* = **polyroots**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*X* = **polyroots**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянный массив:

*X* = **polyroots**([[1,2],[3,4],[5,6]]);

**Результат:**

*X* – выходной массив, содержащий корни характеристического полинома матрицы *М*.

**Пример:**

|  |  |
| --- | --- |
|  | **сonst** M = [[1, -3, 4], [4, -7, 8], [6, -7, 7]];  X = **polyroots**(M); //[-1, -1, 3] |

В результате элементам массива *X* будут присвоены значения [-1, -1, 3], являющиеся корнями характеристического полинома вещественной матрицы *M*.

### Вычитание полиномов

**Синтаксис:**

*С* = **polysubb**(*A, B*);

**Аргументы:**

*A, B* – входные массивы, содержащие коэффициенты a0, a1, …an, b0, b1, …bn полиномов вида:

f(t) = a0+a1t+a2t2+…+antn

g(t) = b0+b1t+b2t2+…+bntn

**Описание:**

*polysub(A, B)* – функция возвращает массив коэффициентов полинома, полученного в результате вычитания из полинома, заданного массивом коэффициентов *А* полинома, заданного массивом коэффициентов *B*.

Входные массивы *A, B* могут задаваться:

 как переменные типа массив, определенные ранее:

*С* = **polysub**(*A, B*);

 как массивы, состоящие из переменных, определенных ранее:

*С* = **polysub**([*a1,a2,a3,a4*],[*b1,b2,b3,b4*]);

 как постоянные массивы:

*С* = **polysub**(([-1, -6, -4, -2],[-4, 7, 5, -3]);

**Результат:**

*С* – выходной массив, содержащий коэффициенты полинома, полученного в результате вычитания из полинома, заданного массивом коэффициентов *А* полинома, заданного массивом коэффициентов *B*:

q(t) = f(t)+g(t) = c0+c1t+c2t2+…+cntn

**Пример:**

|  |  |
| --- | --- |
|  | **const** A = [1, 2, -1, 0, 1, 2];  **const** B = [-1, -1, 2, 1];  сC = **polysub**(A, B); |

В результате элементам массива *C* будут присвоены значения [2, 3, -3, -1, 1, 2], являющиеся коэффициентами полинома q(t) = 2 + 3 t - 3t2 - t3 + t4 + 2t5, полученного в результате вычитания из полинома f(t) = 1 + 2t - t2 + t4 + 2t5 полинома g(t) = -1 – t + 2t2 + t3. Полиномы f(t) и g(t) определенны массивами *A* и *B*.

### Вычисление значения полинома

**Синтаксис:**

*С* = **polyval**(*A, B*);

*x* = **polyval**(*A, y*);

**Аргументы:**

*A* – входной массив, содержащий коэффициенты a0, a1, …an полинома вида:

f(t) = a0+a1t+a2t2+…+antn,

*B* – входной массив, содержащий значения аргументов для вычисления значения полинома,

*y* – значение аргумента для вычисления значения полинома.

**Описание:**

*polyval(A, B)* – функция возвращает массив значений полинома заданного массивом коэффициентов *A*, вычисленных от набора значений аргументов, заданного массивом коэффициентов *B*.

*polyval(A, y)* – функция возвращает значение полинома, заданного массивом коэффициентов *A*, вычисленное от аргумента *y*.

Входные массивы *A, B* могут задаваться:

 как переменные типа массив, определенные ранее:

*С* = **polyval**(*A,B*);

 как массивы, состоящие из переменных, определенных ранее:

*С* = **polyval**([*a1,a2,a3,a4*],[*b1,b2,b3,b4*]);

 как постоянные массивы:

*С* = **polyval**([-1, -6, -4, -2],[-4, 7, 5, -3]);

Аргумент может быть вещественным или комплексным числом.

**Результат:**

*С* – выходной массив, содержащий значения полинома, вычисленные от набора значений аргументов, заданного массивом коэффициентов *B*.

*x* – значение полинома, заданного массивом коэффициентов *A*, вычисленное от аргумента *y*.

**Пример:**

|  |  |
| --- | --- |
|  | **const** A = [1, 2, 3];  C = **polyval**(A, [5, 7, 9]); |

В результате элементам массива C будут присвоены значения [86, 162, 262], являющиеся значениями полинома c(t) = 1 + 2t + 3t2 вычисленными от аргументов 5, 7 и 9 соответственно.

### Вычисление значения, обратного числу обусловленности матрицы

**Синтаксис:**

*x* = **rcond**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*rcond(M)* – функция возвращает значение, обратное числу обусловленности матрицы, основанному на первой норме. Значение также можно получить, используя выражение:



Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*x* = **rcond**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*x* = **rcond**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянная матрица:

*x* = **rcond**([[1,2],[3,4],[5,6]]);

**Результат:**

*х* – значение, обратное числу обусловленности матрицы *M*, основанному на первой норме.

**Пример:**

|  |  |
| --- | --- |
|  | **const** M = [[1, -3, 2], [4, 5, -1], [3, 8, -6]];  x = **rcond**(M); |

В результате переменной *x* будет присвоено значение, обратное числу обусловленности матрицы *М*, равное 0.053125.

### Количество строк матрицы

**Синтаксис:**

*r* = **rows**(*M*);

**Аргументы:**

*M* – входная матрица.

**Описание:**

*rows(M)* – функция возвращает количество строк матрицы.

Входная матрица *М* может задаваться:

 как переменная типа матрица, определенная ранее:

*c* = **rows**(*M*);

 как матрица, состоящая из переменных, определенных ранее:

*x* = **rows**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

 как постоянная матрица:

*x* = **rows**([[1,2],[3,4],[5,6]]);

**Результат:**

*r* – количество строк матрицы *M.* Имеет тип *integer.*

**Пример:**

|  |  |
| --- | --- |
|  | **output** nb; //выход –количество строк матрицы  b = [  [1,2],  [3,4],  [5,6]  ]; //матрица 2x3  nb = **rows**(b); //nb = 3 |

В результате переменной *nb* будет присвоено целое число 3, соответствующее количеству строк матрицы *b*.

### Установка размера матрицы или вектора

**Синтаксис:**

**setsize**(*n*, *X*);

**setsize**(*n*, *m*, *Y*);

**Аргументы:**

*Х* – входной массив или матрица.

*Y* – входная матрица.

*n* – количество элементов или строк результирующего вектора или матрицы,

*m* – количество столбцов результирующей матрицы.

**Описание:**

*setsize(n, X)* – устанавливает количество элементов массива или матрицы как *n* элементов или строк соответственно.

*setsize(n, m, X)* – устанавливает количество строк и столбцов матрицы как *n* и *m* соответственно.

Старые данные сохраняются, новым данным устанавливаются нулевые значения. Если устанавливаемый размер меньше исходного, данные обрезаются.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

**setsize**(*n*, *X*);

 как массив, состоящий из переменных, определенных ранее:

**setsize**(*n*, [*x1,x2,x3,x4*]);

Входная матрица *Y* может задаваться:

 как переменная типа матрица, определенная ранее:

**setsize**(*n*, *m*, *Y*);

 как матрица, состоящая из переменных, определенных ранее:

**setsize**(*n*, *m*, [[*x1,x2*],[*x3,x4*],[*x5,x6*]]);

**Пример 1:**

*Установка нового размера вектора*

|  |  |
| --- | --- |
|  | X = [1,2,3];  **setsize**(10, X); // [1,2,3,0,0,0,0,0,0,0] |

В результате элементам массива *X* будут присвоены значения [1,2,3,0,0,0,0,0,0,0].

**Пример 2:**

*Установка нового количества строк в матрице*

|  |  |
| --- | --- |
|  | X = [[1,2],[3,4]];  **setsize**(5, X); // [[1,2],[3,4],[0,0],[0,0],[0,0]] |

В результате переменной *Х* будут присвоены значения массива [[1,2], [3,4], [0,0], [0,0], [0,0]].

**Пример 3:**

*Установка нового количества строк и столбцов в матрице*

|  |  |
| --- | --- |
|  | Y = [[1]];  **setsize**(2,3, Y); //[[1 , 0 , 0],[0 , 0 , 0]] |

В результате переменной *Y* будут присвоены значения массива [[1 , 0 , 0], [0 , 0 , 0]].

### Сортировка вектора по возрастанию значений

**Синтаксис:**

*Y* = **sort***(X);*

**Аргументы:**

*Х* – входной массив, содержащий элементы вектора.

**Описание:**

*sort(X)* – функция производит сортировку массива по возрастанию элементов. Для комплексных векторов сортировка ведётся по модулям чисел.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*Y* = **sort**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*Y* = **sort**([*x1,x2,x3,x4*]);

 как постоянный массив:

*Y* = **sort**([0,0,2,1,0]);

**Результат:**

*Y* – выходной массив, содержащий элементы отсортированного вектора.

**Пример:**

|  |  |
| --- | --- |
|  | **const** X = [2, 2, 1, 3, 4];  Y = **sort**(X); |

В результате элементам массива *Y* будут присвоены значения [1, 2, 2, 3, 4], представляющие собой отсортированные по возрастанию элементы входного массива.

### Сортировка векторов по возрастанию значений определяющего вектора

**Синтаксис:**

**sortby***(X, Y1, Y2,..,Yn);*

**Аргументы:**

*Х* – входной массив, содержащий элементы сортируемого вектора, определяющего сортировку следующих векторов,

*Y1,..,Yn* – входные массивы, содержащие элементы следующих за определяющим вектором сортируемых векторов.

**Описание:**

*sortby(X, Y1, Y2 ,..,Yn)*– производится сортировка множества векторов по возрастанию значений первого из них (т.е. первый указанный вектор является определяющим, а элементы остальных переставляются в соответствии с порядком перестановки элементов первого). Для комплексных векторов сортировка ведётся по модулям чисел.

Входные массивы *X, Y1, Y2,..,Yn* могут задаваться:

 как переменная типа массив, определенная ранее:

**sortby**(*X, Y1, Y2,..,Yn*);

 как массив, состоящий из переменных, определенных ранее:

**sortby**([*x1,x2,x3,x4*], *Y1, Y2,..,Yn*);

 как постоянный массив:

**sortby**([0,0,2,1,0], *Y1, Y2,..,Yn*);

**Пример 1:**

|  |  |
| --- | --- |
|  | X = [2, 3, 1];  Y = [7, 0, 4];  **sortby**(X, Y); |

В результате элементам массива *Х* будут присвоены значения [1, 2, 3], представляющие собой отсортированные по возрастанию элементы исходного массива *X*, элементам массива *Y* будут присвоены значения [4, 7, 0], представляющие собой отсортированные в соответствии с сортировкой определяющего массива элементы исходного массива *Y*.

**Пример 2:**

|  |  |
| --- | --- |
|  | X = [2, 3, 1];  Y1 = [7, 0, 4];  Y2 = [5, 2, 7];  Y3 = [-1, 0, 8];  **sortby**(X, Y1, Y2, Y3); |

В результате элементам массива *Х* будут присвоены значения [1, 2, 3], представляющие собой отсортированные по возрастанию элементы исходного массива *X*, элементам массивов *Y1, Y2, Y3* будут присвоены значения [4, 7, 0], [7, 5, 2], [8, -1, 0] соответственно, представляющие собой отсортированные в соответствии с сортировкой определяющего массива элементы исходных массивов *Y1, Y2, Y3*.

### Суммирование элементов вещественного или комплексного вектора

**Синтаксис:**

*s* = **sum**(*X*);

**Аргументы:**

*X* – входной массив элементов вектора.

**Описание:**

*sum(X)* – функция возвращает значение суммы всех элементов вектора. Элементы вектора могут быть как вещественными, так и комплексными.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*s* = **sum**(*X*);

 как массив, состоящий из переменных, определенных ранее:

*s* = **sum**([*x1,x2,x3,x4*]);

 как постоянный массив:

*s* = **sum**([1,2,3]);

**Результат:**

*s* – значение суммы всех элементов вектора *X.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** X = [2, 2, 1, 3, 4];  s = **sum**(X); //12 |

В результате переменной *s* будет присвоено значение 12, равное сумме всех элементов массива *X*.

### Транспонирование матрицы или вектора

**Синтаксис:**

*C* = **transp**(*M*);

**Аргументы:**

*M* – массив элементов входной матрицы или вектора.

**Описание:**

*transp(M)* – функция производит транспонирование входной матрицы или вектора *M*.

Входная матрица или вектор *М* может задаваться:

 как переменная типа матрица или вектор, определенная ранее:

*C* = **transp**(*M*);

 как матрица или вектор, состоящие из переменных, определенных ранее:

*C* = **transp**([[*x1,x2*],[*x3,x4*],[*x5,x6*]]);//матрица

*C* = **transp**([*x1,x2*,*x3,x4*,*x5,x6*])//вектор

 как постоянная матрица или массив:

*C* = **transp**([[1,2],[3,4],[5,6]]);//матрица

*C* = **transp**([1,2,3,4,5,6]);//вектор

**Результат:**

*С* – транспонированная матрица для входной матрицы или вектора.

**Пример 1:**

*Транспонирование матрицы:*

|  |  |
| --- | --- |
|  | //массивы элементов матрицы  **const** M = [[1, 0], [-2, 3]];  C = **transp**(M); |

В результате переменной *C* будут присвоены значения массива [[1, -2], [0, 3]], определяющие транспонированную матрицу для матрицы *M* .

**Пример 2:**

*Транспонирование вектора:*

|  |  |
| --- | --- |
|  | //массивы элементов вектора  **const** M = [1, 0, -2, 3];  C = **transp**(M); |

В результате переменной *C* будут присвоены значения матрицы [[1], [0], [-2], [3]], определяющие транспонированную матрицу для вектора *M*.

### Создание нулевого вектора

**Синтаксис:**

*X* = **vector**(*n*);

**Аргументы:**

*n* – количество элементов нулевого вектора.

**Описание:**

*vector(n)* – функция возвращает вектор размерностью n, у которого все элементы равны нулю.

**Результат:**

*X* – возвращаемый массив элементов нулевого вектора*.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** n = 5;  X = **vector**(n); |

В результате переменной *X* будет присвоено значение массива [0, 0, 0, 0, 0], определяющего нулевой вектор, у которой все элементы равны 0.

### Создание вектора единиц

**Синтаксис:**

*X* = **vector1**(*n*);

**Аргументы:**

*n* – количество элементов вектора единиц.

**Описание:**

*vector1(n)* – функция возвращает вектор размерностью n, у которого все элементы равны единице.

**Результат:**

*X* – возвращаемый массив элементов вектора единиц*.*

**Пример:**

|  |  |
| --- | --- |
|  | **const** n = 5;  X = **vector1**(n); |

В результате переменной *X* будет присвоено значение массива [1, 1, 1, 1, 1],

определяющего вектор единиц, у которой все элементы равны 1.

### Вычисление значений скалярной функции

**Синтаксис:**

*Y* = **vectorize**(*t, expr, X*);

**Аргументы:**

*expr* – строка, содержащая цифры и символы ^, \*, +, -, /, определяющие полином вида:

f(t) = a0+a1t+a2t2+…+antn,

*X* – входной массив, содержащий значения аргументов для вычисления значения полинома,

*t* – выходная переменная, с которой определяется полином. Переменной присваивается значение последнего элемента массива *X*.

**Описание:**

*vectorize(t, expr, X)* – функция возвращает массив значений скалярной функции, заданной строкой *expr*, вычисленных от набора значений аргументов, заданного массивом коэффициентов *X*. Выражение *expr* записывается при помощи цифр, буквы аргумента и символов ^, \*, +, -, / и определяет функцию от t, где ^ – возведение аргумента в степень.

Входной массив *X* может задаваться:

 как переменная типа массив, определенная ранее:

*Y* = **vectorize**(*t, expr, X*);

 как массив, состоящий из переменных, определенных ранее:

*Y* = **vectorize**(*t, expr,* [*x1,x2,x3,x4*]);

 как постоянный массив:

*Y* = **vectorize**(*t, expr,* [-1, -6, -4, -2]);

**Результат:**

*Y* – выходной массив, содержащий значения скалярной функции, вычисленные от набора значений аргументов, заданного массивом коэффициентов *X*.

**Пример:**

|  |  |
| --- | --- |
|  | t = 0;  X = [5, 7, 9];  Y = **vectorize**(t, 3\*t^2+2\*t+1, X); |

В результате переменной *Y* будут присвоены значения массива [86, 162, 262], являющиеся значениями скалярной функции c(t) = 1 + 2t + 3t2, вычисленными от аргументов 5, 7 и 9 соответственно. Переменной *t* будет присвоено значение 9.

## Комплексные

### Получение аргумента комплексного числа

**Синтаксис:**

*y* **= arg***(x);*

**Аргументы:**

*x* – входное значение комплексного числа.

**Описание:**

*arg(x)* – функция получения аргумента комплексного числа x. Аргументом комплексного числа называется величина угла, образованного на комплексной плоскости вектором, определяемым комплексным числом, с вещественной осью.

Входное значение *x* может задаваться:

 как переменная типа комплексное число:

*y* = **arg**(*x*);

 как постоянное комплексное число:

*y* = **arg**(3+4i);

**Результат:**

*y* – аргумент комплексного числа. Вещественное число.

**Пример:**

|  |  |
| --- | --- |
|  | y = **arg**(3+4i); |

В результате переменной *y* будет присвоено значение 0.92729522.

### Получение комплексного числа

**Синтаксис:**

*y* **= complex***(a*, *b);*

**Аргументы:**

*a* – вещественная часть комплексного числа,

*b* – мнимая часть комплексного числа,.

**Описание:**

*complex(x)* – функция получения комплексного числа x из его вещественной части *a* и мнимой части *b*.

**Результат:**

*y* – полученное комплексное число. Имеет тип «комплексное число»

**Пример:**

|  |  |
| --- | --- |
|  | y = **complex**(3,4); |

В результате переменной *y* будет присвоено значение 3+4i.

### Получения комплексно-сопряженного числа

**Синтаксис:**

*y* **= conj***(x);*

**Аргументы:**

*x* – входное значение комплексного числа.

**Описание:**

*conj(x)* – функция получения комплексно-сопряженного числа к комплексному числу *x*.

Входное значение *x* может задаваться:

 как переменная типа комплексное число:

*y* = **arg**(*x*);

 как постоянное комплексное число:

*y* = **arg**(3+4i);

**Результат:**

*y* – полученное комплексно-сопряженное число. Имеет тип «комплексное число»

**Пример:**

|  |  |
| --- | --- |
|  | y = **conj**(3+4i); |

В результате переменной *y* будет присвоено значение 3-4i.

### Получение мнимой части комплексного числа

**Синтаксис:**

*y* **= imag***(x);*

**Аргументы:**

*x* – входное значение комплексного числа.

**Описание:**

*imag(x)* – функция получения мнимой части комплексно числа *x*. Функция может быть использована при определении переменных.

Входное значение *x* может задаваться:

 как переменная типа комплексное число:

*y* = **imag**(*x*);

 как постоянное комплексное число:

*y* = **imag**(3+4i);

**Результат:**

*y* – полученное значение. Имеет тип «целое число»

**Пример:**

|  |  |
| --- | --- |
|  | **var**  x:**integer** = **imag**(1+5i);  y = **imag**(3+4i); |

В результате переменной *y* будет присвоено значение 4, переменной *x* будет присвоено значение 5.

### Получение вещественной части комплексного числа

**Синтаксис:**

*y* **= real***(x);*

**Аргументы:**

*x* – входное значение комплексного числа.

**Описание:**

*real(x)* – функция получения вещественной части комплексно числа *x*. Функция может быть использована при определении переменных.

Входное значение *x* может задаваться:

 как переменная типа комплексное число:

*y* = **imag**(*x*);

 как постоянное комплексное число:

*y* = **imag**(3+4i);

**Результат:**

*y* – полученное значение. Имеет тип «целое число»

**Пример:**

|  |  |
| --- | --- |
|  | **var**  x:**integer** = **real**(1+5i);  y = **real**(3+4i); |

В результате переменной *y* будет присвоено значение 3, переменной *x* будет присвоено значение 1.

### Получение комплексного квадратного корня вещественного числа

**Синтаксис:**

*y* **= sqr***(x);*

**Аргументы:**

*x* – входное значение. Имеет тип «целое число»

**Описание:**

*sqr(x)* – функция получения комплексно квадратного корня вещественного числа *x*.

**Результат:**

*y* – полученное значение. Имеет тип «комплексное число»

**Пример:**

|  |  |
| --- | --- |
|  | y = **sqr**(16); |

В результате переменной *y* будет присвоено значение 4+0i.

## Гиперболические

Таблица 3.5 – Гиперболические функции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Sh** | **sh**(<аргумент>) | Гиперболический синус действительного или комплексного числа |
| **Ch** | **ch**(<аргумент>) | Гиперболический косинус действительного или комплексного числа |
| **Th** | **th**(<аргумент>) | Гиперболический тангенс действительного или комплексного числа |
| **Arcsh** | **arcsh**(<аргумент>) | Гиперболический арксинус действительного или комплексного числа |
| **Arch** | **arcch**(<аргумент>) | Гиперболический арккосинус действительного или комплексного числа |
| **Arcth** | **arcth**(<аргумент>) | Гиперболический арктангенс действительного или комплексного числа |

## Тригонометрические

Таблица 3.6 – Тригонометрические функции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Sin** | **sin**(<аргумент>) | Синус действительного или комплексного числа |
| **Cos** | **cos**(<аргумент>) | Косинус действительного или комплексного числа |
| **Tg** | **tg**(<аргумент>) | Тангенс действительного или комплексного числа |
| **Ctg** | **ctg**(<аргумент>) | Котангенс действительного или комплексного числа |
| **Arctg** | **arctg**(<аргумент>) | Арктангенс действительного или комплексного числа |
| **Arcsin** | **arcsin**(<аргумент>) | Арксинус действительного или комплексного числа |
| **Arcos** | **arccos**(<аргумент>) | Арккосинус действительного или комплексного числа |

## Интерполяция

Таблица 3.7 –Функции интерполяции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Cspline** | **cspline**(X,Y) | Функция вычисляет матрицу коэффициентов для кубической сплайн-интерполяции по заданным координатам точек. Вычисленная матрица коэффициентов используется функцией *interpol* для сплайн-интерполяции аргумента *x*. Входные массивы X и Y должны быть одного размера. |
| **Interpol** | **interpol**(x,M)  **interpol**(X,Y,argument)  **interpol**(X,Y,Z, argX,argY) | *interpol(x, M)* – функция кубической cплайн-интерполяции аргумента *x* по вычисленной ранее при помощи функции *cspline* матрице коэффициентов *M*. Вычисление производится согласно формуле:  где i – индекс интервала  *interpol(X,Y, х)* – функция двумерной линейной интерполяции аргумента *x* по заданным координатам точек.  *interpol(X,Y,Z, х, y)* – функция трехмерной линейной интерполяции аргументов *x,y* по заданным координатам точек. |
| **Linterp** | *M* = **linterp**(*X*, *Y*); | Функция вычисляет матрицу коэффициентов для линейной интерполяции по заданным координатам точек. Вычисленная матрица коэффициентов используется функцией *interpol* для линейной интерполяции аргумента *x*.  Входные массивы X и Y должны быть одного размера. |
| **Mnkpoly** | *P* = **mnkpoly**(*X, Y, S, n*); | *mnkpoly(A, B)* – функция возвращает массив коэффициентов полинома степени *n*, полученного в результате аппроксимации таблично-заданной функции *Y*(*X*), с заданным среднеквадратичным отклонением *S* для каждой точки. Вычисление производится при помощи сингулярного матричного преобразования (SVD). Если значения среднеквадратического отклонения не известны, их можно задавать равными единице. |

## Специальные

Таблица 3.8 – Специальные функции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **besseli0** | **besseli0**(<аргумент>) | Модифицированная функция Бесселя I нулевого порядка |
| **besseli1** | **besseli1**(<аргумент>) | Модифицированная функция Бесселя I первого порядка |
| **besselI** | **Besseli**(<порядок>,<аргумент>) | Модифицированная функция Бесселя I порядка больше чем 2. **Порядок всегда должен быть скалярным.** |
| **besselj0** | **besselj0**(<аргумент>) | Модифицированная функция Бесселя J нулевого порядка |
| **besselj1** | **besselj0**(<аргумент>) | Модифицированная функция Бесселя J первого порядка |
| **besselj** | **besselj**(<порядок>,<аргумент>) | Модифицированная функция Бесселя J порядка больше чем 2. **Порядок всегда должен быть скалярным.** |
| **besselk0** | **besselk0**(<аргумент>) | Модифицированная функция Бесселя K нулевого порядка |
| **besselk1** | **besselk0**(<аргумент>) | Модифицированная функция Бесселя K первого порядка |
| **besselk** | **besselk** (<порядок>,<аргумент>) | Модифицированная функция Бесселя K порядка больше чем 2. **Порядок всегда должен быть скалярным.** |
| **bessely0** | **bessely0**(<аргумент>) | Модифицированная функция Бесселя Y нулевого порядка |
| **bessely1** | **bessely0**(<аргумент>) | Модифицированная функция Бесселя Y первого порядка |
| **bessely** | **bessely**(<порядок>,<аргумент>) | Модифицированная функция Бесселя Y порядка больше чем 2. **Порядок всегда должен быть скалярным.** |
| **gamma** | **gamma**(<аргумент>) | Полная Гамма-функция |
| **gammln** | **gammln**(<аргумент>) | Логарифм полной Гамма-функции |
| **gammp** | **gammp**(a,x) | Частичная Гамма-функция P(a,x). Если аргументы – матрицы или ветора, то их размерности должны совпадать. |
| **gammq** | **gammq**(a,x) | Частичная Гамма-функция Q(a,x). Если аргументы – матрицы, то их размерности должны совпадать. |
| **erf** | **erf**(x) | Функция ошибок Гаусса определяется как    Функция вычисляет функцию ошибок Гаусса от аргумента *x*. |
| **erfc** | **erfc**(x) | Дополнительная функция ошибок Гаусса определяется через функцию ошибок Гаусса    Функция вычисляет дополнительную функцию ошибок Гаусса от аргумента *x*. |

## Свойства воды и водяного пара

Таблица 3.9 – Свойства воды и водяного пара

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **waterps** | **waterps**(P, flag\*) | Специальная функция, вычисляющая свойства **воды** по **давлению** на **линии насыщения**. Диапазон: давления от 0.09 до 50 МПа и температуры от 10 до 800 °С. |
| **waterts** | **waterts**(T, flag\*) | Специальная функция, вычисляющая свойства **воды** по **температуре** на **линии насыщения**. Диапазон: давления от 0.09 до 50 МПа и температуры от 10 до 800 °С. |
| **waterpt** | **waterpt**(P, T, flag\*) | Специальная функция, вычисляющая свойства **воды** по **давлению** и **температуре**. Диапазон: давления от 0.09 до 50 МПа, температуры от 10 до 800°С. |
| **waterph** | **waterph**(P, H, flag\*) | Специальная функция, вычисляющая свойства **воды** по **давлению** и **энтальпии**. Диапазон: давления от 0.09 до 50 МПа, температуры от 10 до 800°С. |
| **steamps** | **steamps**(P, flag\*) | Специальная функция, вычисляющая свойства **водяного пара** по **давлению** на **линии насыщения**. Диапазон: давления от 0.09 до 50 МПа и температуры от 10 до 800 °С. |
| **steamts** | **steamts**(P, flag\*) | Специальная функция, вычисляющая свойства **водяного пара** по **давлению** на **линии насыщения**. Диапазон: давления от 0.09 до 50 МПа и температуры от 10 до 800 °С. |
| **steampt** | **steampt**(P, T, flag\*) | Специальная функция, вычисляющая свойства **водяного пара** по **давлению** и **температуре**. Диапазон: давления от 0.09 до 50 МПа, температуры от 10 до 800 °С. |
| **steamph** | **steamph**(P, H, flag\*) | Специальная функция, вычисляющая свойства **водяного пара** по **давлению** и **энтальпии**. Диапазон: давления от 0.09 до 50 МПа, температуры от 10 до 800°С. |

Значения параметра **flag** могут изменяться от 1 до 12 и соответствовать вычислению следующих термодинамических характеристик **водяного пара**:

**1** – давление;

**2** – температура;

**3** – энтальпия;

**4** – удельный объем;

**5** – число Прандтля;

**6** – динамическая вязкость;

**7** – коэффициент теплопроводности;

**8** – энтропия;

**9** – удельная теплоёмкость Cp;

**10** – удельная теплоёмкость Cv;

**11** – производная плотности по энтальпии при постоянном давлении (∂ρ/∂i)P;

**12** – производная плотности по давлению при постоянном объёме (∂ρ/∂p)V;

Входные параметры и возвращаемые значения термодинамических параметров водяного пара представляются в системе СИ (за исключением температуры, которая измеряется в °С).

## Статистические

Таблица 3.10 – Статистические функции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Fft** | **fft**(X) | Вычисляет прямое дискретное преобразование Фурье вектора X методом быстрого преобразования Фурье. **Размер вектора должен быть степенью 2.** |
| **Ifft** | **ifft**(X) | Вычисляет обратное дискретное преобразование Фурье вектора X методом быстрого преобразования Фурье. **Размер вектора должен быть степенью 2.** |
| **Moment3** | **moment3**(X) | Возвращает значение 3-го момента вектора X (коэффициент эксцесса). |
| **Moment4** | **moment**(X) | Возвращает значение 4-го момента вектора X (фактор сплющиваемости). Для распределения Гаусса равен 3. |
| **Mean** | **mean**(X) | Вычисление среднего значения элементов действительного вектора X. |
| **Rms** | **rms**(X) | Вычисление среднеквадратичного отклонения элементов действительного вектора X. |

## Функции случайных чисел

Таблица 3.11 – Функции случайных чисел

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Rand** | **Rand** | Генератор равномерного шума в диапазоне значений [0, 1]. |
| **Randg** | **randg**(M,S) | Генератор Гуссовского шума с математическим ожиданием M и дисперсией S. |
| **Randp** | **randp**(M) | Генератор случайных целых чисел с распределением Пуассона с математическим ожиданием M. |
| **Randgamma** | **randgamma**(P) | Генератор шума с Гамма-распределением порядка P. |

## Геометрические

Таблица 3.12 – Геометрические функции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **center** | **center**(*P1, P2, P3*) | Функция вычисления координат центра окружности, заданной тремя точками *P1*, *P2*, *P3*. Входные значения могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **color** | **color**(*x*, *min\_x, max\_x, min\_c, max\_c*) | Функция вычисления линейной интерполяция цвета, где аргумент *x* заключён в интервале от *min\_x* до *max\_x*, а цвет варьируется в интервале от *min\_c* до *max\_c*. |
| **distance** | **distance**(*P1, P2)* | Функция вычисления расстояния между точками *P1*, *P2*. Входные значения могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **excenter** | **excenter**(*P1, P2, P3*) | Функция вычисления расстояния от точки *P3* до прямой, заданной точками *P1, P2*. Входные значения могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **getangle** | **getangle**(*P1, P2*) | Функция определения угла между полярной осью и вектором, заданным точками *P1*, *P2* в декартовой системе координат. Угол определяется в радианах. |
| **Middle** | **middle**(*P1, P2*) | Функция вычисления расстояния между точками *P1*, *P2*.  Входные значения могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **slide** | **slide**(*P1, P2, Ps, Pe, l*) | Функция вычисления координат точки, принадлежащей прямой, заданной точками *P1*, *P2* и удаленной от заданной точки *Ps* на заданное расстояние *l*. Точка *Pe* определяет расположение вычисляемой точки. |
| **xcoord** | **xcoord**(*P*) | Функция получения координаты по оси абсцисс (координата x) точки *P*. Входные значения могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **ycoord** | **ycoord**(*P*) | Функция получения координаты по оси ординат (координата y) точки *P*. Входные значения могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |

## Файловые

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Savetab** | **savetab**(M,”имя файла”) | Сохранение матрицы в текстовый файл. Матрица будет сохранена в транспонированном виде, т.е. строкам матрицы будут соответствовать столбцы в текстовом файле. |
| **Loadtab** | **loadtab**(“имя файла”) | Загрузка матрицы из текстового файла. Матрица загружается в транспонированном виде, т.е. столбцам текста соответствуют строки матрицы. |
| **createfile** | **FileID = createfile(**"имя файла",идентификатор открытия**)** | Создать файл. Значения идентификатора: -1 – новый файл, 0 - чтение, 1 - запись, 2 - чтение и запись. |
| **freeobject** | **freeobject(**FileID**)** | Уничтожить объект (файл) |
| **seek** | **seek(FileId,Position)** | Установить текущую позицию в файле |
| **getpos** | **getpos(**FileId**)** | Получить текущую позицию в файле |
| **filesize** | **filesize(**FileId**)** | Получить размер файла |
| **writefile** | **writefile(**FileID,var1,var2 ...**)** | Записать данные в файл |
| **writetext** | **writetext(**FileID,var1,var2 ...**)** | Записать данные в файл в текстовом виде |
| **writeln** | **writeln(**FileID,var1,var2 ...**)** | Записать в текстовом виде с переводом строки |
| **readfile** | **readfile(**FileID,var1,var2 ...**)** | Записать данные в файл |
| **copyfile** | **copyfile**(src, dest)  **copyfile**(src, dest, flag) | *copyfile(src, dest)* – функция производит копирование файла с именем *src* в файл с именем *dest*.  *copyfile(src, dest, flag)* – функция производит копирование файла с именем *src* в файл с именем *dest*.  Если *flag* равен 1, то в случае, если файл с именем *dest* уже существует, он не будет переписан |
| **fileexist** | **fileexists**(name) | Функция определяет наличие файла с именем *name*. |
| **findfile** | **findfiles**(name, attr, s\_id) | *findfiles(name, attr, s\_id)* – функция поиска файлов по фильтру, заданному в строке *name*, с атрибутами *attr*. Имена найденных файлов возвращаются в списке строк с идентификатором *s\_id*.  Атрибуты файлов могут принимать значения:  0- Файлы "только чтение". Такой атрибут устанавливается на файлы, которые не рекомендовано изменять, удалять. Такой атрибут имеют файлы, например, записанные на компакт-дисках.  1- Скрытые файлы. При обычных установках браузера и командира эти файлы невидимы.  2- Системные файлы.  3- Файл метки диска. Такой элемент в своем имени имеет название диска (максимум 11 символов).  4- Атрибут признака каталога.  5- Обычный файл. По умолчанию устанавливается на заново создаваемых файлах.  6- Если установить в качестве атрибута искомых элементов, то будет произведен поиск по всем вышесказанным атрибутам. |
| **loadtext** | **loadtext**(f\_id) | Функция чтения текста из файла с идентификатором *f\_id* в строку |
| **readln** | **readln**(f\_id) | Функция чтения одной строки из текстового файла с идентификатором *f\_id* в строку. Если строк в файле больше нет, возвращает пустую строку, положение указателя файла можно проверить функцией *getpos* и *filesize*. |

## Работа с памятью

Таблица 3.14 – Функции работы с памятью

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **~** | **~**<тип данных>(<имя переменной>)**;** | Доcтуп к переменной заданного типа по ссылке ptr. Например:  var i=10,arr:array;  ptr=@i; //получение ссылки  ptr1=@arr;  d=~double(ptr); //доступ по ссылке  w=~array(ptr1);  ~double(ptr)=1000; |
| **new~** | **new~<**тип данных> | Динамическое выделение памяти и присвоение ссылки. Например:  p=new~double; //Выделение памяти  ~double(p) = 100; //Присвоение  y=~double(p); //Чтение  dispose(p); //Освобождение памяти |
| **dispose** | **dispose(**<имя ссылки>) | Освобождение памяти, выделенной по ссылке ptr динамически при помощи new~… |

## Строковые

Таблица 3.15 – Функции строковые

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **addstringlist** | **addstringlist**(s\_id, str, tag) | Функция добавления строки с меткой *tag* в конец списка строк с идентификатором *s\_id*. |
| **addtohash** | **addtohash**(h\_id, n) | Добавление элемента списка строк (строки) с номером *n* в хэш-таблицу с идентификатором *h\_id*.  Хэш-таблицы предназначены для обеспечения ускорения поиска идентификатора в списке строк. |
| **assignstringlist** | **assignstringlist**(src\_id, dest\_id) | Копируется содержимое списка строк с идентификатором *src\_id* в список строк с идентификатором *dest\_id.* |
| **clearstringlist** | **clearstringlist**(s\_id) | Функция очищает список строк с идентификатором *s\_id*. |
| **countstringlist** | **countstringlist**(s\_id) | Функция возвращает количество строк в списке строк с идентификатором *s\_id*. |
| **createstringlist** | **createstringlist** | Функция создания списка строк и получения на него идентификатора. Список строк можно уничтожить при помощи *freeobject*. |
| **createstringlisthash** | **createstringlisthash**(s\_id, mem) | Функция создания хэш-таблицы для списка строк с идентификатором *s\_id* и получения ее идентификатора. |
| **deletestringlist** | **deletestringlist**(s\_id, indx) | Производится удаление строки с индексом (номером) *indx* из списка строк с идентификатором *s\_id*. Нумерация строк в списке начинается с нуля. |
| **getindexesfromhash** | **getindexesfromhash**(h\_id, str) | Функция осуществляет поиск индексов строки *str* в списке строк в соответствии с хэш-таблице *h\_id*. |
| **getindexfromhash** | **getindexfromhash**(h\_id, str); | Функция осуществляет поиск индекса строки *str* в списке строк в соответствии с хэш-таблице *h\_id*. |
| **getstringlist** | **getindexfromhash**(h\_id, str) | Функция получения строки с номером *indx* из списка строк с идентификатором *s\_id*. Нумерация строк в списке начинается с нуля. |
| **gettagstringlist** | **getstringlist**(s\_id, indx) | Функция получения метки строки по номеру строки *indx* из списка строк с идентификатором *s\_id*. Нумерация строк в списке начинается с нуля. |
| **gettextstringlist** | **gettagstringlist**(s\_id, indx) | Функция преобразовывает список строк с идентификатором *s\_id* в строку, строки из списка разделяются разделителем «перевод строки». |
| **indexofstringlist** | **indexofstringlist**(s\_id, str) | Функция определяет номер строки *str* в списке строк с идентификатором *s\_id*. Строки в списке нумеруются с нуля. |
| **insertstringlist** | insertstringlist(s\_id, indx, str, tag) | Функция добавления строки с меткой *tag* в список строк с идентификатором *s\_id*, строка добавляется на место с заданным номером *indx*, нумерация начинается с нуля. |
| **loadstringlistfromfile** | **loadstringlistfromfile**(s\_id, filename) | В список строк с идентификатором *s\_id* загружаются строки из файла, содержащего текст, строки в котором разделяются разделителями «конец строки» или «перевод каретки». |
| **removefromhash** | **removefromhash**(h\_id, str) | Удаление строкового идентификатора *str* из хэш-таблицы с идентификатором *h\_id*. Данная операция удаляет все записи в хэш-таблице, содержащие строку *str.* |
| **savestringlisttofile** | **savestringlisttofile**(s\_id, filename) | Строк строк с идентификатором *s\_id* сохраняется в файл. В результате файл содержит текст, строки из списка разделяются разделителями «конец строки» или «перевод каретки». |
| **setstringlist** | **setstringlist**(s\_id, indx, str, tag) | Функция вставки строки *str* с меткой *tag* на позицию с номером *indx* в список строк с идентификатором *s\_id*. Нумерация строк в списке начинается с нуля. Строка вставляется вместо строки на позиции *indx.* Возвращается номер позиции вставленной строки. |
| **settagstringlist** | **gettagstringlist**(s\_id, indx, tag) | Функция установки метки строки *tag* по номеру строки *indx* в списке строк с идентификатором *s\_id*. Нумерация строк в списке начинается с нуля. |
| **settextstringlist** | **settextstringlist**(s\_id, str) | В список строк с идентификатором *s\_id* загружаются строки из строки, содержащей текст, разделенный разделителями «конец строки» или «перевод каретки». |
| **addzeros** | **addzeros**(str\_src, count); | Функция добавления нулей в начало строки, количество знаков строки увеличивается до количества *count*. |
| **asc** | **asc**(str); | Функция возвращает ASCII-код первого символа строки. Если строка пустая, то возвращает 0. |
| **chr** | **chr**(c); | Функция возвращает строку с символом, имеющим ASCII-код, равный *c*. |
| **datatostr** | **datatostr**(x); | Функция формирует строку из переменной произвольного типа. |
| **floattostr** | **floattostr**(x); | Функция формирует строку из переменной *x* типа «число с плавающей запятой». Переменная *х* должна быть типа *double*. |
| **floattostrf** | **floattostrf**(x, type, par1, par2) | Функция формирует строку заданного формата из переменной *x* типа «число с плавающей запятой». Переменная *х* должна быть типа *double*. Переменная *type* определяет формат вывода:  1.Формат автоматический (general).  2.Формат научный (scientific).  3.Формат фиксированный (fixed). |
| **format** | **format**(type, par1, par2) | Установка формата формирования строки из переменной типа «число с плавающей запятой». Формат используется в функции datatostr и при отображении переменных в окне просмотра переменных. Если количество цифр в формируемой строке меньше количества цифр исходного числа, число округляется.  Переменная *type* определяет формат вывода. |
| **getdbfileasis** | **getdbfileasis**(dbfilename); | Функция возвращает имя файла базы данных в формате хранения - т.е. Возвращает частичный путь к файлу бд без добавления в начало полного пути к папке database |
| **getitem** | **getitem**(str\_t, indx); | Функция получения строки (фрагмента текста) с номером *indx* из строки, содержащей текст, разделенный разделителями «конец строки» или «перевод каретки». Нумерация фрагментов в тексте начинается с единицы. |
| **getitemscount** | **getitemscount**(str\_t); | Функция возвращает количество фрагментов текста в строке, содержащей текст, разделенный разделителями «конец строки» или «перевод каретки». |
| **gettextsize** | **gettextsize**(str, font); | Функция получения размера строки *str*, отображаемой шрифтом *font*.  Тип возвращаемого значения - комплексное число, где действительная часть соответствует ширине строки, мнимая - высоте. |
| **gettoken** | **gettoken**(str, n); | Функция получения части строки *str*, начиная с символа с номером *n*. Нумерация символов в строке начинается с единицы. При этом аргумент *n* меняет своё значение на позицию в строке для следующей лексемы (слова). Пробелы и переносы игнорируются. |
| **length** | **length**(str); | Функция возвращает количество символов в строке. Если строка пустая, то возвращает 0. |
| **lowercase** | **lowercase**(str); | Функция переводит все символы строки в нижний регистр. Работает только с латинскими символами. |
| **pos** | **pos**(sub\_str, str); | – Функция поиска части строки *sub\_str*, в строке *str*. Если часть строки найдена, возвращается номер символа в строке, начиная с которого расположена часть строки. Нумерация символов в строке начинается с единицы. Если возвращаемое значение меньше 1, то подстрока не найдена. |
| **setitem** | **setitem**(str\_t, indx, str); | Функция замены строки (фрагмента текста) с номером *indx* в строке, содержащей текст, разделенный разделителями «конец строки» или «перевод каретки». Замена происходит на строку str. Нумерация фрагментов в тексте начинается с единицы. |
| **strcopy** | **strcopy**(str, indx, count); | Функция копирования части строки размером *count* символов из строки *str*, начиная с номера символа *indx*. Нумерация символов в строке начинается с единицы. Скопированная часть строки возвращается в строку *sub\_str*. |
| **translit** | **translit**(str); | Функция переводит кириллические символы строки в латинские символы (производит транслитерацию строки). |
| **uppercase** | **uppercase**(str); | Функция переводит все символы строки в верхний регистр. Работает только с латинскими символами. |
| **valf** | **valf**(str, c); | Функция формирует переменную типа «число с плавающей запятой», заданную строкой *str*. Функция возвращает код ошибки в аргумент *c* целого типа, если строковое значение является не числом, или 0 в случае успеха операции. |
| **vali** | **vali**(str, c); | Функция формирует переменную типа «целое число», заданную строкой *str*. Функция возвращает код ошибки в аргумент *c* целого типа, если строковое значение является не числом, или 0 в случае успеха операции. |

## Функции времени

Таблица 3.16 – Функции времени

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **aperiodika** | **aperiodika**(u, time, T)  **aperiodika** (u, T) | *Aperiodika*(*u, time, T*) – функция вычисления апериодического звена 1-го порядка входной величины *u* с постоянной времени звена *T* и текущим модельным временем *time*.  *Aperiodika*(*u, T*) – функция вычисления апериодического звена 1-го порядка входной величины *u* с постоянной времени звена *T*. |
| **datetimetostr** | **datetimetostr**(date) | Функция преобразования значения даты и времени в системном формате из переменной *date* типа *double* в строку. Формат времени в строке определяется системными установками |
| **datetimetounix** | **datetimetounix**(date\_s) | Функция преобразования значения даты и времени в системном формате (переменная типа *double*) в формат UNIX (вещественное число секунд начиная с 01.01.1970). Системный формат времени в строке определяется системными установками. |
| **dayof** | **dayof**(date) | Функция извлечения значения дня из значения даты и времени в системном формате. |
| **flash** | **flash**(time) | Производит дискретное изменение значения на выходе с 0 на 1 с заданным периодом *time* в миллисекундах. Изменение выходного значения привязано к системному времени. Может использоваться для отображения мигающих элементов. |
| **hourof** | **hourof**(date) | Функция извлечения значения часа из значения даты и времени в системном формате. |
| **millisecondof** | **millisecondof**(date) | Функция извлечения значения миллисекунды из значения даты и времени в системном формате. |
| **minuteof** | **minuteof**(date) | Функция извлечения значения минуты из значения даты и времени в системном формате. |
| **monthof** | **monthof**(date) | Функция извлечения значения месяца из значения даты и времени в системном формате |
| **now** | **now** | Функция получения текущего значения даты и времени в системном формате. |
| **secondof** | **secondof**(date) | Функция извлечения значения секунды из значения даты и времени в системном формате |
| **sleep** | **sleep**(time) | Здержка выполнения операций на заданное число миллисекунд. Эту функцию можно использовать при выполнении асинхронных операций, для уменьшения загрузки процессора при наличии там циклов. |
| **step** | **flash**(time, val1, val2) | Функция производит дискретное изменение значения на выходе с *val1* на *val2* через время *time*. Функция доступна в блоке "язык программирования" |
| **tickcount** | time = **tickcount** | Функция определения времени, прошедшего с момента запуска операционной системы в миллисекундах. Может использоваться для подсчёта реальных временных интервалов. |
| **unixtodatetime** | **unixtodatetime**(date\_u) | Функция преобразования значения даты и времени в формате UNIX (вещественное число секунд начиная с 01.01.1970) в системный формат (переменная типа *double*). Системный формат времени в строке определяется системными установками. |
| **yearof** | **yearof**(date) | Функция извлечения значения года из значения даты и времени в системном формате |

## Графические и системные

Таблица 3.17 – Графические и системные функции

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **Графики** | | |
| **getgraphicid** | **getgraphicid**(*gr\_name*) | Функция получения значения идентификатора объекта типа «График» на схеме по имени *gr\_name*. Идентификатор имеет тип указателя на объект. Имя свойства не обрамляется кавычками |
| **getgraphiclegend** | **getgraphiclegend**(*gid*) | Функция получения свойства «Показывать легенду» объекта типа «График» по идентификатору объекта. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **getgraphicmousevalue** | **getgraphiсmousevalue**(*gid, num*) | Функция получения координаты текущей точки на графике в соответствии с указателем мыши на схеме по идентификатору объекта *gid* и номеру шкалы *num*. Указатель мыши определяет координату по оси x, для которой получается координата y графика. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. Номер шкалы соответствует номеру графика (начиная с 0) при многошкальном режиме отображения. Если используется одна шкала, то номер шкалы можно указывать произвольно. Тип возвращаемого значения - комплексное число, где действительная часть соответствует значению по оси абсцисс, мнимая - по оси ординат. |
| **getgraphicxinterval** | **getgraphicxinterval**(*gid*) | Функция получения значения свойства «Интервал видимости по оси X» объекта типа «График» по идентификатору объекта *gid*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **getgraphicxscales** | **getgraphicxscales**(*gid*) | Функция получения значения свойств «Минимум шкалы по оси X» и «Максимум шкалы по оси X» объекта типа «График» по идентификатору объекта *gid*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. Тип возвращаемого значения *–* комплексное число, где действительная часть соответствует минимальному значению шкалы, мнимая *–* максимальному. |
| **getgraphicyinterval** | **getgraphicyinterval**(*gid, num*) | Функция получения значения свойства «Интервал видимости по оси Y» объекта типа «График» по идентификатору объекта *gid* и номеру шкалы num. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. Номер шкалы соответствует номеру графика (начиная с 0) при многошкальном режиме отображения. Если используется одна шкала, то номер шкалы можно указывать произвольно. |
| **getgraphicyscales** | **getgraphicxscales**(*gid, num*) | Функция получения значения свойств «Минимум шкалы по оси Y» и «Максимум шкалы по оси Y» объекта типа «График» по идентификатору объекта *gid* и номеру шкалы *num*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. Номер шкалы соответствует номеру графика (начиная с 0) при многошкальном режиме отображения. Если используется одна шкала, то номер шкалы можно указывать произвольно. Тип возвращаемого значения *–* комплексное число, где действительная часть соответствует минимальному значению шкалы, мнимая *–* максимальному. |
| **resetgraphic** | **resetgraphic**(*gid*) | Функция сброса показаний объекта типа «График» по идентификатору объекта *gid*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphiclegend** | **setgraphiclegend**(*gid, fl*) | Функция установки свойства «Показывать легенду» объекта типа «График» по идентификатору объекта. Аргумент *fl* принимает значения 1 – показывать легенду или 0 – не показывать легенду. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicxauto** | **setgraphicxauto**(*gid, fl*) | Функция установки свойства «Автомасштабирование по оси X» объекта типа «График» по идентификатору объекта. Аргумент *fl* принимает значения:   1 – установить автомасштабирование по оси X;   0 – убрать автомасштабирование по оси X.  Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicxgrid** | **setgraphicxgrid**(*gid, fl*) | Функция установки свойства «Сетка по оси X» объекта типа «График» по идентификатору объекта. Аргумент *fl* принимает значения:   1 – установить сетку по оси X;   0 – убрать сетку по оси X.  Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicxinterval** | **setgraphicxinterval**(*gid, val*) | Функция установки значения *val* свойства «Интервал видимости по оси X» объекта типа «График» по идентификатору объекта *gid*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicxscales** | **setgraphicxscales**(*gid, val*) | Функция установки значения val свойств «Минимум шкалы по оси X» и «Максимум шкалы по оси X» объекта типа «График» по идентификатору объекта *gid*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. Тип аргумента *val* *–* комплексное число, где действительная часть соответствует минимальному значению шкалы, мнимая *–* максимальному. Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и мнимые части числа соответственно. |
| **setgraphicyauto** | **setgraphicyauto**(*gid, fl*) | Функция установки свойства «Автомасштабирование по оси Y» объекта типа «График» по идентификатору объекта. Аргумент *fl* принимает значения:   1 – установить автомасштабирование по оси Y;   0 – убрать автомасштабирование по оси Y.  Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicygrid** | **setgraphicygrid**(*gid, fl*); | Функция установки свойства «Сетка по оси Y» объекта типа «График» по идентификатору объекта. Аргумент *fl* принимает значения:   1 – установить сетку по оси Y;   0 – убрать сетку по оси Y.  Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicyinterval** | **setgraphicyinterval**(*gid, val*) | Функция установки значения *val* свойства «Интервал видимости по оси Y» объекта типа «График» по идентификатору объекта *gid*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. |
| **setgraphicxycales** | **setgraphicxscales**(*gid, num, val*) | Функция установки значения val свойств «Минимум шкалы по оси Y» и «Максимум шкалы по оси Y» объекта типа «График» по идентификатору объекта gid и номеру шкалы *num*. Идентификатор объекта может быть получен функцией *getgraphicid(gr\_name)* и имеет тип указателя на объект. Номер шкалы соответствует номеру графика (начиная с 0) при многошкальном режиме отображения. Если используется одна шкала, то номер шкалы можно указывать произвольно. Тип аргумента *val* *–* комплексное число, где действительная часть соответствует минимальному значению шкалы, мнимая *–* максимальному. Комплексное число задается выражением *a*+*b*i, где *a* и *b* вещественные и мнимые части числа соответственно. |
| **Графические** | | |
| **bringtofront** | **bringtofront**(*name*); | Функция перемещения имени объекта в списке отрисовки в конец списка, то есть объект будет отрисовываться на схеме последним. |
| **сonstr2p** | **const2p**(*P1*, *P2*, *name1, name2,…, namen*) | Функция привязки объектов с именами *name1, name2,…,namen* к точкам *P1* и *P2* на схеме. Входные значения *P1* и *P2* могут задаваться как заранее определенные переменные или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **createprimitiv** | **createprimitiv**(*type*, [*P1*, *P2*,…, *Pn*]) | Функция динамического создания графического объекта на схеме типа *type*, расположение графического объекта задается координатами *P1*, *P2*,…, *Pn*. Количество аргументов (*n*), задающих координаты, зависит от типа объекта *type*. Количество координат для объекта определенного типа можно определить, поставив вручную на схему объект этого типа и скопировав его координаты из свойств объекта. |
| **global** | **global**(*Pl*); | Функция вычисления глобальных координат на схеме по локальным координатам графического контейнера. |
| **loadbitmap** | **loadbitmap**(*img, filename*) | Функция загрузки изображения в сигнал или свойство объекта на схеме с именем *img* из графического файла *filename*. |
| **move** | **move**(P, *name1, name2,…, namen*) | Функция перемещения объектов на схеме с именами *name1, name2,…,namen* на вектор, заданный координатами *P*. Входное значение *P* может задаваться как заранее определенная переменная или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. *P* – переменная типа *point*, где x – расстояние для перемещения по оси X, y – расстояние для перемещения по оси Y. |
| **removeprimitiv** | **removeprimitiv**(*gid*) | Функция удаление объекта со схемы по его идентификатору *gid*. |
| **restore** | **restore**(*name1, name2,…, namen*) | Функция сброса координат объектов с именами *name1, name2,…, namen* на схеме. |
| **rgbtocolor** | **rgbtocolor**(*r, g, b*) | Функция получения кода цвета по его отдельным компонентам. Код цвета вычисляется при помощи формулы r\*2^0 + g\*2^8 + b\*2^16, или r + g\*256 + b\*65536 |
| **rotate** | **rotate**(*P*,*a*,*name1, name2,…, namen*) | Функция поворота объектов на схеме с именами *name1, name2,…, namen* на угол *a* относительно центра, заданного координатами *P*. Угол поворота задается в радианах. Входное значение *P* может задаваться как заранее определенная переменная или задаваться выражением (*x*, *y*), где *x* и *y* координаты точки. |
| **savescreenshot** | **savescreenshot**(*filename, type*) | Функция сохранения в графический файл с именем *filename* текущего изображения экрана проекта в формате, определяемом кодом *type*. |
| **scale** | **scale**(*P*,*s*,*name1, name2,…, namen*) | Функция масштабирования объектов на схеме с именами *name1, name2,…, namen* относительно центра, заданного координатами *P* с коэффициентом *s*. |
| **setfontcolor** | **setfontcolor**(*font, val*); | Функция установки значения кода цвета val шрифта для объекта на схеме. Аргумент *val* должен быть числом и не может быть переменной. |
| **setfontname** | **setfontname**(*font, name*) | Функция установки типа шрифта с названием *name* для объекта на схеме |
| **setfontprop** | **setfontprop**(*name, par, val*) | Функция установки значения цвета или размера шрифта для объекта с именем *name* на схеме. При установке значения цвета шрифта аргумент *par* задается строкой “*color”*, при установке значения размера шрифта аргумент *par* задается строкой “*size”.* |
| **setfontsize** | **setfontsize**(*font, val*) | Функция установки размера *val* шрифта для объекта на схеме. |
| **setfontstyle** | **setfontstyle**(*font, val*) | Функция установки маски *val* стиля шрифта для объекта на схеме. Значения битовых масок *val* для стилей шрифта: 1 – жирный; 2 – курсив; 4 – подчёркнутый; 8 – зачёркнутый. |
| **setvisible** | **setvisible**(*name, fl*) | Функция установки свойства видимости для объекта с именем *name* на схеме. Аргумент *fl* принимает значения 1 – объект видим или 0 – объект не видим. |
| **slideline** | **slideline**(*P1*, *P2*, *Ps*, *name*) | Функция привязки линейного объекта с именем *name* к точке на схеме *Ps* и линии, заданной точками *P1* и *P2*. Начальная точка объекта с именем *name* привязывается к свободной точке *Ps*, конечная точка привязывается к линии, заданной точками *P1* и *P2* |
| **storeposition** | **storeposition**(*name1, name2,…, namen*) | Функция запоминания координат объектов с именами *name1, name2,…, namen* на схеме. |
| **stretch** | **stretch**(*P*, *S*, *name1, name2,…, namen*) | Функция масштабирования объектов на схеме с именами *name1, name2,…,namen* относительно центра, заданного координатами *P* с коэффициентами, определяемыми вектором *S*. *S* – переменная типа *point*, где x – коэффициент растяжения/сжатия по оси X, y – коэффициент растяжения/сжатия по оси Y. |
| **zoomrect** | **zoomrect**(*left*, *top*, *right, bottom*) | Функция увеличения фрагмента изображения на схеме, заданного прямоугольной областью, ограниченной границами *left*, *top*, *right, bottom*. При увеличение может происходить подгонка фрагмента изображения под заданную область, т.к заданная прямоугольная область не всегда пропорционально соответствует ширине и высоте фрагмента изображения |
| **Менеджер данных** | | |
| **addengineparam** | **addengineparam**(*en*\_*id, en\_name, par\_name, path*) | Функция добавления параметра или сигнала *par\_name* из блока *path* в параметр или сигнал *en\_name* объекта по ссылке *en\_id*, находящегося в менеджере данных. Объект в менеджере данных – устройство отображения или ввода информации (графики, регистраторы, панели управления и т.п.). Ссылка на объект в менеджере данных возвращается функцией *findengine*. |
| **addparamlogaddeddata** | **addparamlogaddeddata**(*par*\_*id, ust\_reg, ust\_prior, ust\_val, ust\_descr*) | Функция установки данных уставки (режим *ust\_reg*, приоритет *prior*, значение *ust\_val*, описание *ust\_descr*) для параметра по ссылке *par\_id* объекта «Регистратор событий», находящегося в менеджере данных. Ссылка параметра *par\_id* *–* целое число, возвращаемое функцией *addengineparam*. Объект в менеджере данных – устройство отображения или ввода информации (графики, регистраторы, панели управления и т.п.). |
| **changeengineparams** | **changeengineparams**(*id*) | Функция инициализации списка параметров объекта по его ссылке *id*, находящегося в менеджере данных. Объект в менеджере данных – устройство отображения или ввода информации (графики, регистраторы, панели управления и т.п.). Ссылка имеет тип указателя на объект. Ссылка на объект в менеджере данных возвращается функцией *findengine*. |
| **clearengineparams** | **clearengineparams**(*id*) | Функция очистки списка параметров объекта по его ссылке *id*, находящегося в менеджере данных. Объект в менеджере данных – устройство отображения или ввода информации (графики, регистраторы, панели управления и т.п.) Ссылка имеет тип указателя на объект. Ссылка на объект в менеджере данных возвращается функцией *findengine*. |
| **closeengine** | **closeengine**(*en\_name*) | Функция осуществляет закрытие окна объекта *–* устройства отображения или ввода информации (графики, регистраторы, панели управления и т.п.) С именем *en\_name*, находящегося в менеджере данных. Имя объекта обрамляется кавычками. |
| **findengine** | **findengine**(*en\_name*) | Функция получения ссылки на объект – устройство отображения или ввода информации (графики, регистраторы, панели управления и т.п.) С именем *en\_name*, находящийся в менеджере данных. Ссылка имеет тип указателя на объект. Имя объекта обрамляется кавычками. |
| **gotoengine** | **gotoengine**(*en\_name*) | Функция осуществляет переход на объект *–* устройство отображения или ввода информации (графики, регистраторы, панели управления и т.п.) С именем *en\_name*, находящееся в менеджере данных. Имя объекта обрамляется кавычками. |
| **loaddatamanager** | **loaddatamanager**(*filename*) | Функция загрузки конфигурации окна менеджера данных из файла с именем *filename*. Имя файла должно быть с расширением .mgr. |
| **resetengines** | **resetengines** | Функция осуществляет сброс состояния объектов *–* устройств отображения или ввода информации (графики, регистраторы, панели управления и т.п.), находящихся в менеджере данных. |
| **showcontrolform** | **showcontrolform**(*name, control:wind\_name*) | Функция отображения окна управления с именем *wind\_name* объектом с именем *name*. |
| **Мышь** | | |
| **getmousepos** | *P* = **getmousepos** | Функция получения координаты указателя мыши в системе координат основного графического контейнера (окна редактора). Тип возвращаемого значения - комплексное число, где действительная часть соответствует значению по оси абсцисс, мнимая - по оси ординат. |
| **getwheeldelta** | *s* = **getwheeldelta** | Функция получения величины последнего перемещения колеса мыши для активного графического окна. После вызова функции величина перемещения автоматически сбрасывается в 0. |
| **moiseincontainer** | *fl* = **mouseincontainer** | Функция получения флага нахождения курсора мыши в переделах графического контейнера (группы или блока). Возвращает true, если курсор находится в пределах данного графического контейнера. |
| **setmousepos** | **setmousepos**(*x*, *y*) | Функция установки указателя мыши в точку с координатами *x*,*y* в системе координат основного графического контейнера (окна редактора). |
| **setmousetocenter** | **setmousetocenter** | Функция установки указателя мыши в центре окна текущего графического контейнера (вызвавшего данную функцию). |
| **Рестарты (исходные состояния)** | | |
| **readprojectrestart** | **readprojectrestart**(*filename*) | Функция считывания точки рестарта проекта из файла с именем *filename* |
| **writeprojectrestart** | **writeprojectrestart**(*filename*) | Функция сохранения точки рестарта проекта в файл с именем *filename* |
| **writerestartpoint** | **writerestarttopoint** | Функция сохранения точки рестарта проекта в файл с именем, заданным в настройках проекта. Имя файла задается в закладке "рестарт" |
| **Сигналы** | | |
| **addsignaltolist** | **addsignaltolist***(type, reg, name, descr, val, init*) | Функция добавляет сигнал с режимом *reg,* именем *name,* названием (описанием) *descr* и начальным значением *val* в список сигналов проекта. |
| **clearsignallist** | **clearsignallist***(num*) | Функция удаляет список сигналов проекта, начиная с сигнала с номером *num*. Сигналы нумеруются с нуля. |
| **signalexist** | *flag* = **signalexist***(name*) | Функция определяет наличие сигнала с именем *name* в проекте. Возвращает значение логической 1, если сигнал (переменная) в данном проекте доступен. |
| **Системные** | | |
| **changeother** | **changeother***(win\_id, prt\_file, name\_id, layer\_id, lcmd\_id, layer\_data, start\_fl, resize\_fl*) | Функция закрывает текущий проект и загружает новый проект из файла *prt\_file* в заданном окне с идентификатором *win\_id* в асинхронном режиме (сразу возвращает управление). Параметр *win\_id* является опциональным. Функция возвращает 1, если функция выполнена успешно. Если флаг запуска проекта *start\_fl* равен 1, то загруженный проект сразу запускается. Если флаг установки новых размеров *resize\_fl* равен 0, то размеры окна не изменятся. |
| **changeproject** | **changeproject***(prt\_file, name\_id, layer\_id, lcmd\_id, layer\_data, start\_fl, resize\_fl*) | Функция загружает и открывает проект из файла *prt\_file* в асинхронном режиме (сразу возвращает управление). Функция возвращает идентификатор открытого проекта. Если флаг *start\_fl* установлен равным 1, то в случае, если проект уже открыт, активизируется его окно. Если флаг установки новых размеров *resize\_fl* равен 0, то размеры окна не изменятся. |
| **changeprojectzoom** | **changeprojectzoom***(prt\_file, name\_id, layer\_id, lcmd\_id, layer\_data, start\_fl, resize\_fl, left, top, right, bottom*) | Функция загружает и открывает проект из файла *prt\_file* в асинхронном режиме (сразу возвращает управление) , а также производит масштабирование изображения в соответствии с прямоугольной областью, ограниченной границами *left*, *top*, *right, bottom*. При увеличение может происходить подгонка фрагмента изображения под заданную область, т.к заданная прямоугольная область не всегда пропорционально соответствует ширине и высоте фрагмента изображения. Функция возвращает идентификатор открытого проекта. Если флаг *start\_fl* установлен равным 1, то в случае, если проект уже открыт, активизируется его окно. Если флаг установки новых размеров *resize\_fl* равен 0, то размеры окна не изменятся. |
| **closeapp** | **closeapp** | Функция завершает работу приложения. |
| **closeform** | **closeform** | Функция закрывает текущее окно, на котором отображается видеокадр или схема |
| **doneobject** | **doneobject**(*obj\_name*) | Функция принудительно деинициализирует объект по его имени *obj\_name* |
| **execmethod** | **execmethod**(*name*) | Функция выполняет метод главного окна приложения с названием *name*. В случае успешного выполнения возвращает 1, иначе 0. |
| **execpropscript** | **execpropscript**(*blok\_name*, *prop\_name*) | Функция выполняет скрипт свойства с названием *prop\_name* блока сидентификатором *block\_name*. В случае успешного выполнения возвращает 1, иначе 0. |
| **exitapp** | **exitapp** | Функция завершает работу приложения из скрипта. |
| **findindexerkey** | **findindexerkey***(key, filtr\_name, filtr\_type, sprj\_id, sblkpath\_id, sblktype\_id, spropname\_id, fl\_find*) | Функция поиска ключевого слова *key* с фильтрами имен свойств *filtr\_name* и типов блоков *filtr\_type* в индексаторе проектов. Результаты поиска возвращаются в аргументах *sprj\_id –* идентификатор списка строк с именами найденных проектов, *sblkpath\_id –* идентификатор списка строк с путями найденных блоков в проекте, *sblktype\_id –* идентификатор списка строк с типами найденных блоков в проекте, *sprop\_id –* идентификатор списка строк с найденными свойствами блоков в проекте. |
| **findobjectbyfullname** | **findobjectbyfullname***(name*) | Функция выполняет поиск объекта по заданному полному пути *name* в проекте. |
| **findobjectbyname** | **findobjectbyname***(name*) | Функция выполняет поиск объекта по заданному имени *name* в проекте |
| **findobjectbynametrans** | **findobjectbynametrans***(name*) | Функция выполняет поиск объекта по заданному имени *name* с транслитерацией в проекте. |
| **getformbounds** | **getformbounds**(*left*, *top*, *width*, *height*) | Функция получения положения (левая *left* и верхняя *top* граница) и размеров (ширина *width* и высота *height*) окна отображения текущего графического контейнера. Отсчёт координат по горизонтали начинается с левого края текущего монитора вправо, по вертикали - с верхнего края текущего монитора вниз. Параметры данной функции являются возвращаемыми и должны быть обязательно типа integer. |
| **getfullname** | **getfullname**(*obj\_name*) | Функция получения полного пути объекта в проекте по заданному имени *obj\_name*. |
| **gethwnd** | **gethwnd** | Функция получения оконного контекста (идентификатора) окна редактора |
| **getobj** | **getobj**(*indx*) | Функция получения идентификатора (ссылки) объекта (блока на схеме или другого графического объекта) по индексу. Индекс начинается с единицы. Идентификатор имеет тип указателя на объект. |
| **getobjclassname** | **getobjclassname**(*obj\_id*) | Функция получения имени типа объекта в графическом контейнере по его идентификатору *obj\_id*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. |
| **getobjcount** | **getobjcount** | Функция получения количества объектов в графическом контейнере. |
| **getobjectparent** | **getobjectparent**(*obj\_id*) | Функция получения идентификатора родительского объекта для объекта в графическом контейнере по его идентификатору *obj\_id*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. |
| **getobjectvisuallayer** | **getobjectvisuallayer**(*num*, *obj\_id*) | Функция получения флага наличия визуального слоя c номером *num* для объекта в графическом контейнере по его идентификатору *obj\_id*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Номер слоя задается в диапазоне от 1 до 16 |
| **getobjname** | **getobjname**(*obj\_id*) | Функция получения имени объекта в графическом контейнере по его идентификатору *obj\_id*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. |
| **getprojectenabledlayer** | **getprojectenabledlayer**(*num*) | Функция получения флага чувствительности визуального слоя c номером *num* проекта. Номер слоя задается в диапазоне от 1 до 16. |
| **getprojectvisiblelayer** | **getprojectvisiblelayer**(*num*) | Функция получения флага видимости визуального слоя c номером *num* проекта. Номер слоя задается в диапазоне от 1 до 16. |
| **getprop** | **getprop**(*id*,*"ob\_name"*) | Функция получения значения свойства объекта на схеме по имени свойства. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой, обрамляется кавычками. |
| **getpropasstring** | **getpropasstring**(*id*,*"ob\_name"*) | Функция получения текстового значения свойства объекта с идентификатором *id* на схеме по имени свойства *"ob\_name"*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой, обрамляется кавычками. |
| **getpropevalstring** | **getpropevalstring**(*id*,*"ob\_name"*) | Функция получения интерпретированного текста (формулы), содержащего свойство по имени *"ob\_name"* объекта с идентификатором *id* . Возвращаемое значение типа строка. Для не интерпретируемых типов данных возвращает текстовое значение, аналогично getpropasstring. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой, обрамляется кавычками. |
| **getstrformat** | **getstrformat**(*format*\_*str*) | Функция получения интерпретированного текста по строке, содержащей текст в установленном формате, аналогично форматированию, используемому в свойствах блоков. Возвращаемое значение типа строка. |
| **getsubstring** | **getsubstring**(*str, sep\_str, index*) | Функция получения подстроки в строке с разделителями. Функция возвращает текст из исходной строки, от разделителя с указанным номером до следующего разделителя. |
| **getsystemvar** | **getsystemvar**(*name*\_*str*) | Функция получения строки со значением, установленным ранее при помощи функции *setsystemvar,* системной переменной с именем *name\_str*. |
| **getwindowbycomid** | **getwindowbycomid**(*com\_id*) | Функция получения идентификатора окна проекта по его уникальному COM-идентификатору. Если окно есть – результат не нулевой. |
| **getwindowid** | **getwindowid**(*num*) | Функция получения идентификатора окна проекта по его номеру. Нумерация начинается с нуля. Если окно есть – результат не нулевой. |
| **gotopage** | **gotopage**(*name*) | Функция осуществляет переход на страницу с заданным именем в схемном окне. Имя страницы задается строкой (обрамляется кавычками). |
| **initobject** | **initobject**(*obj\_name*) | Функция осуществляет принудительную переинициализацию объекта по его имени *obj\_name.* |
| **iseditprops** | *flag* = **iseditprops** | Функция определяет нахождение одного из свойств блоков в режиме редактирования*.* Возвращает 1, если производится редактирование свойств блоков. |
| **ismainpage** | *flag* = **ismainpage** | Функция определяет нахождение исполняемого скрипта на главной странице проекта. Возвращает 1, если скрипт находится на главное странице проекта, 0 *–* если скрипт находится на странице субмодели. |
| **isrun** | *flag* = **isrun** | Функция определяет нахождения мнемосхемы в режиме отображения. Возвращает 1, если мнемосхема в режиме отображения |
| **messagebox** | **messagebox**(*str*, *caps*, *type*)  **messagebox**(*str*, *caps*, *type*, *par\_id*) | *Messagebox(str, caps, type)* – функция выводит системное окно сообщения типа *type* с заголовком *caps*, текстом *str*.  *Messagebox(str, caps, type, par\_id)* *–* функция выводит системное окно сообщения типа *type* с заголовком *caps*, текстом *str*. *Par\_id* *–* идентификатор родительского окна сообщения. |
| **openproject** | *prt\_id* = **openproject***(prt\_file, start\_fl*) | Функция загружает и открывает проект из файла *prt\_file*. Функция возвращает идентификатор открытого проекта. Если флаг *start\_fl* установлен равным 1, то в случае, если проект уже открыт, активизируется его окно. |
| **popupmenu** | **popupmenu***(left\_point, menu\_str*) | Функция создаёт и выводит на экран всплывающее меню с положением левой верхней точки left\_point и текстом *menu\_str*, определяющим конфигурацию меню. Функция возвращает управление сразу и не производит ожидания, пока меню не будет выбрано |
| **prjinited** | *flag* = **prjinited** | Функция определяет инициализацию текущего проекта на расчет. Возвращает 1, если текущий проект был проинициализирован на расчёт. |
| **reinitsubmodel** | **reinitsubmodel**(*obj\_id*) | Функция производит компиляцию скриптов и переинициализацию для выбранной субмодели с идентификатором *obj\_id.* |
| **reloadblockimage** | **reloadblockimage***(obj\_name, gcn\_file*) | Функция загружает изображение объекта типа "группа" или "блок" с именем *obj\_name* из указанного файла с именем *gcn\_file* (файлы графических контейнеров с расширением .gcn). В случае использования данной функции внутри скрипта свойства блока, в качестве имени своего блока надо использовать ключевое слово self |
| **removeimagefromcash** | **removeimagefromcash***(img\_file*) | Функция производит выгрузку изображения, загруженного ранее из файла с именем *img\_file,* из кэша картинок программы. Функция используется для принудительного освобождения места в кэше при смене изображения в блоке "картинка из файла". Перед вызовом функции необходимо в скрипте сменить изображение, переназначив свойство imagefile данного примитива и вызвав функцию initobject. |
| **removeobject** | **removeobject**(*obj\_name*) | Функция удаляет объект с именем *obj\_name* со схемы |
| **setblocklabel** | **setblocklabel**(*obj\_name, str*) | Функция устанавливает текст *str* подписи для прямоугольного блока с именем *obj\_name*, расположенного на схеме*.* |
| **seterrorflag** | **seterrorflag**(*error\_fl*) | Функция устанавливает флаг принудительной ошибки инициализации расчёта |
| **seterrorstr** | **seterrorstr**(*error\_str, error\_type*)  **seterrorstr**(*error\_str, error\_type, obj\_id*) | *Seterrorstr(error\_str, error\_type)* *–* функция вывода сообщения типа *error\_type* с текстом *error\_str* в окно сообщений проекта*.*  *Seterrorstr(error\_str, error\_type, obj\_id)* *–* функция вывода сообщения типа *error\_type* с текстом *error\_str* в окно сообщений проектаc идентификатор объекта *obj\_id* для его указания по двойному щелчку |
| **setformbounds** | **setformbounds**(*left*, *top*, *width*, *height*) | Функция установки положения (левая *left* и верхняя *top* граница) и размеров (ширина *width* и высота *height*) окна отображения текущего графического контейнера. Отсчёт координат по горизонтали начинается с левого края текущего монитора вправо, по вертикали - с верхнего края текущего монитора вниз. |
| **setformposition** | **setformposition**(*left*, *top*) | Функция установки положения (левая *left* и верхняя *top* граница) окна отображения текущего графического контейнера. Отсчёт координат по горизонтали начинается с левого края текущего монитора вправо, по вертикали - с верхнего края текущего монитора вниз. |
| **setformsize** | **setformsize**(*width*, *height*) | Функция установки размеров (ширина *width* и высота *height*) окна отображения текущего графического контейнера. |
| **sethintautohide** | **sethintautohide**(*flag*) | Функция устанавливает флаг *flag* (1-установить флаг, 0- сбросить флаг) автоматического скрытия всплывающих подсказок в окне редактора. |
| **setobjectvisuallayer** | **setobjectvisuallayer**(*num*, *flag, obj\_name1, …, obj\_namen*) | Функция установки флага *flag* (включен - 1 или выключен - 0 ) наличия визуального слоя c номером *num* для объектов в графическом контейнере по их именам *obj\_name1… obj\_namen*. Номер слоя задается в диапазоне от 1 до 16. |
| **setpainttimer** | **setpainttimer**(*time*) | Функция устанавливает интервал перерисовки *time* (в миллисекундах) для текущего окна редактора или окна управления. |
| **setpointcount** | **setpointcount**(*obj\_name, num*) | Функция устанавливает количество *num* точек графического примитива с именем *obj\_name* с переменным количеством точек (полилинии, многоугольника и т.п.) |
| **setprojectenabledlayer** | **setprojectenabledlayer** (*num, flag*) | Функция получения флага *flag* (включен – 1 или выключен – 0) чувствительности (активности) визуального слоя c номером *num* проекта. Номер слоя задается в диапазоне от 1 до 16. |
| **setprojectvisiblelayer** | **setprojectvisiblelayer**(*num, flag*) | Функция установки флага видимости визуального слоя *flag* (включен - 1 или выключен - 0 ) c номером *num* проекта. Номер слоя задается в диапазоне от 1 до 16 |
| **setprop** | **setprop**(*id*,*"ob\_name"*,*var*) | Присваивание значения свойству объекта на схеме. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой (обрамляется кавычками). Значение, присваиваемое свойству объекта, должно иметь тип свойства объекта |
| **setpropasstring** | *Flag* = **setpropasstring**(*id*,*"ob\_name", str*) | Функция установки текстового значения, содержащегося в строке *str* свойства объекта с идентификатором *id*на схеме по имени свойства*"ob\_name"*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой, обрамляется кавычками. |
| **setpropevalstring** | *flag* = **setpropevalstring**(*id*,*"ob\_name", str*) | Функция установки интерпретированного текста (формулы) *str*, содержащего значение свойства по имени *"ob\_name"* для объекта с идентификатором *id*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой, обрамляется кавычками. |
| **setpropnamebycaption** | **setpropnamebycaption**(*obj*\_*id*,*"ob\_name"*, *new\_name*) | Функция изменяет имя свойства *ob\_name* объекта с идентификатором *id* на схеме на имя *new\_name*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Текущее и новое имя свойства задается строкой (обрамляется кавычками). |
| **setpropvaluebycaption** | **setpropvaluebycaption**(*obj*\_*id*,*"ob\_name"*, *val\_str*) | Функция изменяет значение свойства *ob\_name* объекта с идентификатором *id* на схеме на новое текстовое значение *val\_str*. Идентификатор объекта может быть получен функцией *getobj(i)* и имеет тип указателя на объект. Имя свойства задается строкой (обрамляется кавычками). |
| **setsignalevalstring** | *flag* = **setsignalevalstring**(*"sig\_name", str*) | Функция устанавливает интерпретированный скрипт (формулу) сигнала с именем *"sig\_name"* графического контейнера, в скрипте которого используется данная формула. Имя сигнала задается строкой, обрамляется кавычками. Возвращаемое значение равно 1, если функция успешно присвоила свойство, и 0, если этого выполнить не удалось. |
| **setsystemvar** | **setsystemvar**(*name*\_*str, var\_str*) | Функция установки системной переменной *name\_str* значения, определяемого строкой *var\_str*. |
| **setwindowcaption** | **setwindowcaption**(*name\_str*) | Функция установки заголовка окна редактора. |
| **shellexec** | **shellexec***(op, file, param, path*) | Функция выполняет произвольную команду оболочки ОС Windows. |
| **showgraphicprintdialog** | **showgraphicprintdialog**(*obj\_id*) | Функция выводит окно печати графика с идентификатором *obj\_id*, полученным при помощи функции *getgraphicid*. |
| **startsecondproject** | **startsecondproject***(prt\_file, start\_fl, bd\_fl*) | Функция загружает и открывает вторичный проект, находящийся в файле *prt\_file*, из текущего проекта. Если флаг *start\_fl* установлен равным 1, то в случае, если проект уже открыт, активизируется его окно. Если флаг *bd\_fl* установлен равным 1, то происходит объединение базы сигналов запускаемого проекта с базой сигналов текущего. |

## Функции доступа к БД через ADO

Таблица 3.18 – Функции доступа к БД через ADO

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **adoqueryclose** | **adoqueryclose(<id объекта доступа к БД>)** | Закрыть подключение к БД |
| **adoquerycreate** | db\_id = **adoquerycreate;** | Создать объект для доступа к БД через ADO, возвращает id объекта доступа к БД. Для уничтожения объекта вызовите freeobject |
| **adoqueryfirst** | **adoqueryfirst**(<id объекта доступа к БД>**)** | Установить первую запись как текущую |
| **adoquerygetdataasstring** | data = **adoquerygetdataasstring**(<id объекта доступа к БД>,<номер столбца начиная с 0>) | Получить данные из запроса в строковом виде по номеру столбца |
| **adoquerynext** | **adoquerynext**(<id объекта доступа к БД>**)** | Перейти на следующую запись. |
| **adoqueryprocesssql** | **adoqueryprocesssql**(<id объекта доступа к БД>,"текст SQL запроса к БД"**)** | Произвести SQL - запрос к БД в соответствии с ранее заданной строкой подключения. Если запрос не обработан - возвращает false |
| **adoqueryrecordcount** | recs = **adoqueryrecordcount**(<id объекта доступа к БД>**)** | Получить к-во данных на выходе запроса |

Продолжение таблицы 3.15

|  |  |  |
| --- | --- | --- |
| **adoquerysetconnection** | **adoquerysetconnection**(<id объекта доступа к БД>, "строка подключения к БД") | Установить строку подключения к БД. |

## Функции прямого доступа к интерфейсу обмена с удаленной системой

Таблица 3.19 – Функции доступа к БД через ADO

|  |  |  |
| --- | --- | --- |
| **Функция** | **Параметры** | **Назначение** |
| **createextmodule** | obj\_context=**createextmodule("**[имя\_DLL@имя\_объекта](mailto:имя_DLL@имя_объекта)**")** | Создать внешний объект из DLL из скрипта |
| **dbginitialize** | **dbginitialize**(<obj\_context>,"параметры соединения","имя объекта",<целое возвращаемое - хэш>) | Инициализировать соединение с интерфейсом доступа |
| **dbgaddreadvar** | **dbgaddreadvar** (<obj\_context>,DATA,"имя переменной в исполнительной системе") | Добавить переменную в канал обмена для чтения |
| **dbgaddwritevar** | **dbgaddwritevar**(<obj\_context>,DATA,"имя переменной в исполнительной системе",<флаг однократной записи = 0>) | Добавить переменную в канал обмена для записи, |
| **dbgstart** | **dbgstart**(<obj\_context>) | Старт соединения |
| **dbgexchange** | **dbgexchange**(<obj\_context>) | Обмен данными |
| **dbgsaverestart** | **dbgsaverestart**(<obj\_context>,"имя рестарта") | Сохранить точку рестарта на удалённой исполнительной системе |
| **dbgloadrestart** | **dbgloadrestart**(<obj\_context>,"имя рестарта") | Загрузить точку рестарта на удалённой исполнительной системе |