Основы языка программирования ABB RAPID

Оглавление

1. ROBOSTUDIO RAPID PROGRAMMING LANGUAGE MANUAL

В данном разделе будут рассмотрены основы языка программирования RAPID в ABB RobotStudio.

2. ПЕРЕМЕННЫЕ

a. Основные типы данных

Тип данных	Описание
num	Числовые значения. Может содержать как натуральные числа, так и десятичные. Пример 10 или 3.1456. Хранит только 4 знака после запитой.
dnum	Числовые значения с более высокой точностью чем num. Может содержать как натуральные числа очень и десятичные. Пример 811231238 или 5.12301905
string	Строка, например, “Hello !!!”. Содержит символьные значения. Размер строки не больше чем 80 символов.
bool	Булевские значения. Может хранить только TRUE или FALSE

Объявление переменной происходит с указание типа производной и имени производной. Перед объявлением переменной указывается ключевое слово VAR. Пример:

VAR «data type» «varialbe name»; - Ключевое слово VAR «тип данных» «имя переменной».

```
VAR num speed;
VAR dnum radius;
VAR string status;
VAR bool isStarted;
```

Присваивается значение переменной, используется команда :=. Обратите внимание что знак := это не знак равенства, а знак присваивание значения.

Пример:

variable name := value – имя переменой := значение

```
speed:= 26;
radius := 55.33;
status := "All ok";
isStarted := FALSE;
```

Значение переменной можно задавать при её объявлении. Пример:

VAR «data type» «variable name» := value - VAR «тип данных» «имя переменной» := значение.

```
VAR bool isStarted := FALSE;
```

NB!!! Язык программирования RAPID имеет статическую типизацию – это значит, что тип переменной задаётся при объявлении переменой. Если у переменной числовой тип, то присвоение значение другого типа недопустимо. Если всё-таки идёт присвоение переменой значение другого типа, то среда разработки выдаст ошибку и подчеркнёт то место где это происходит «Рисунок 1».

```
VAR num speed;
speed := "23.33";
         Type mismatch. Expected num
```

Рисунок 1 Ошибка присваивание значения

b. Сохраняющиеся переменные

В языке программирования RAPID, как и во многих других, значение переменной обнуляется после прохода программы. В RAPID существуют переменные, которые сохраняют своё значение после выполнения программы. Они называются **«Сохраняющиеся переменные (Persistent variables)».** В отличии от простых переменных сохраняющиеся переменные объявляются ключевым словом **PERS**. Пример:

PERS «data type» «variable name» := value – PERS «тип данных» «имя переменной» := значение.

```
PERS num number := 3;
```

Рассмотрим пример простой программы.

Создадим переменную строкового типа NowIS и присвоим ей значение “Day”. В блоке main() присвоим к переменой NowIS значение “Night”. И запустим программу.

```
PERS string NowIS := "Day";
PROC main()
    NowIS:="Night";
ENDPROC
```

После выполнения программы к переменной NowIS присвоится значение “Night”. И в код программы автоматически изменится.

```
PERS string NowIS := "Night";
PROC main()
    NowIS:="Night";
ENDPROC
```

Как можно заметить значение переменной NowIS изменилось и после выполнения программы значение перенеслось в объявление переменной.

c. Константа.

Константа содержит значение, так же, как и переменная, но значение всегда назначается в объявлении переменой и после этого значение не может быть изменено. Константа может быть использована в программе так же, как переменная, за исключением того, что к переменной нельзя присвоить новое значение.

Объявление константы происходит с помощью ключевого слова CONST. Пример:

CONST «data type» «variable name» := value – CONST «тип переменной» «имя переменной» := значение

```
CONST string sayHello := "Hello";
CONST num length := 451.33;
```

3. ОПЕРАТОРЫ

a. Числовые операторы

Числовые операторы работают только с числами и результат выражение тоже число. В таблице num1, num2 и num3 переменные числового типа (num или dnum).

Оператор	Описание	Пример
+	Сложение	`num3:=num1+num2;`
-	Вычитание	`num3:=num1-num2;`
*	Умножение	`num3:=num1*num2;`
/	Деление	`num3:=num1/num2;`

b. Операторы отношения

Операторы отношений работают с разными типами данных, но результат их работы является булевские значение (TRUE или FALSE). В таблице num1 и num2 – числовые типы, result – булевское.

Оператор	Описание	Пример
=	Равно	result: = num1 = num2; result будет True если num1 и num2 будут равны
<	Меньше	result: = num1 < num2; result будет True если num1 меньше чем num2
>	Больше	result: = num1 > num2; result будет True если num1 больше чем num2
<=	Меньше или равно	result: = num1 <= num2; result будет True если num1 меньше чем num2 или равны
>=	Больше или равно	result: = num1 >= num2; result будет True если num1 больше num2 или равны
<>	Не равно	result: = num1 < > num2; result будет True если num1 и num2 не будут равны

c. Конкатенация

Конкатенация - операция склеивания объектов линейной структуры, обычно строк. Например, конкатенация слов «микро» и «мир» даст слово «микромир».

Пример RAPID кода.

```
1.  VAR string firstname: = "Иван";
2.  VAR string lastname: = "Иванов";
3.  VAR string fullname;
4.  fullname: = firstname + " " + lastname;
```

Переменная `fullname` будет равна «Иван Иванов»

4. ЛОГИЧЕСКИЕ ИНСТРУКЦИИ IF/ELSE

Логические инструкции нужны, для того, чтобы проверять условия в коде и выполнять команды по условию. Например, нужно выполнить определённую команду по условию, тогда используем инструкцию IF, если условие в операторе истинно, тогда будет выполняться код записанный между IF и ENDIF. А если ложно тогда блок команд, записанный в оператор IF не будет выполнен, и программа продолжит своё выполнение за пределами этого блока.

Пример:

Допустим нам нужно проверить пустая ли строка. Тогда код будет такой.

```
1. VAR string string1: = "Hello";
2. IF string1 < > ""
3. THEN TPWrite string1;
4. ENDIF
```

В этой программе мы видим, что создаётся строка со значением «Hello» и затем проверяется не равна ли она пустой строке, если не ровна, то выполнится условие, записанное между IF и ENDIF. То есть выполнится команда «THEN TPWrite string1;».

Логическая инструкция ELSE выполняется, тогда, когда условие, записанное в IF, является ложью.

```
1. VAR string string1: = "Hello";
2. IF string1 < > ""
3. THEN TPWrite string1;
4. ELSE TPWrite "The string is empty";
5. ENDIF
```

В этой программе видно, что если строка пустая, то выполнится команда, записанная после оператора ELSE «TPWrite "The string is empty"; ».

Если нужно проверить несколько условий, тогда можно использовать инструкцию ELSEIF. Пример:

Если нужно проверить быстро ли была сделана деталь.

```
1. VAR num time: = 38.7;
2. IF time < 40 THEN TPWrite "Part produced at fast rate";
3. ELSEIF time < 60 THEN TPWrite "Part produced at average rate";
4. ELSE TPWrite "Part produced at slow rate";
5. ENDIF
```

В этом примере показано если деталь сделана со скоростью меньше чем 40 секунд, то выполнится команда «`TPWrite "Part produced at fast rate";`», если будет в промежутке между 40 и 60 секундами, то выполнится команда «`TPWrite "Part produced at average rate";`», а если будет больше 60 секунд, то выполнится команда «`TPWrite "Part produced at slow rate";`»

Если нужно проверить одновременно несколько условий тогда можно использовать ключевые слова OR и AND. Пример:

```
1.  VAR num temp: = 15;
2.  IF temp <= 40 OR temp >= 15 THEN TPWrite "Cool";
3.  ELSEIF temp > 40 THEN TPWrite "Hot";
4.  ELSE TPWrite "Cold";
5.  ENDIF
```

Если температура будет в пределах 15-40, то будет выполнена команда «TPWrite "Cool";» если будет больше 40 то выполнится команда «THEN TPWrite "Hot";», в остальных случаях выполнится команда «TPWrite "Cold";».

5. ЦИКЛ FOR И WHILE

Во многих случаях требуется повторить какую-нибудь операцию несколько раз подряд. Это возможно сделать несколькими способами. Самый простой это несколько раз подряд записать команду или набор команд, что увеличит размер кода в разы и читабельность кода уменьшится. Чтобы этого избежать были придуманы циклы. В языке программирования RAPID существует два цикла For и While, каждый из них предназначен для своей цели. Например, цикл FOR повторит команду (блок команд) несколько раз подряд с определённым шагом.

Пример цикла For:

Допустим нужно выписать на экран строку 7 раз. Например, можно написать команду несколько раз подряд.

```
TPWrite("Hello");
TPWrite("Hello");
TPWrite("Hello");
TPWrite("Hello");
TPWrite("Hello");
TPWrite("Hello");
TPWrite("Hello");
```

Результатом этой программы будет вывод на экран 7 раз строки «Hello». Однако, если нужно будет изменить количество команд с 7 до 100, тогда уже будет солоновато написать такую программу копирование кода.

Если эту же задачу выполним с использованием цикла FOR, тогда результат будет такой.

```
FOR i FROM 1 TO 7 DO
    TPWrite("Hello");
ENDFOR
```

Как можно заметить количество строк в коде уменьшилось и читабельность увеличилась.

Рассмотрим конструкцию цикла FOR. Начало цикла начинается с команды FOR, затем создаётся переменная i (итератор), которая при каждом проходе цикла будет менять своё значение. Затем идёт условие цикла с чего начинать (FROM) до (TO) какого момента заканчивать выполнять процедуру. Затем идёт команда выполнить (DO), после которой идёт тело цикла, которое должно выполнится. ENDFOR указывает на конец цикла.

Также бывают задачи выполнить определённую операцию до какого-то условия. Например, нужно посчитать с 1 до 100. Для этого предусмотрен цикл WHILE.

```
VAR num sum := 0;
VAR string answer;
WHILE sum<>100 DO
    sum:=sum+1;
    answer := NumToStr(sum,0);
    TPWrite(answer);
ENDWHILE
```

В этом примере показана работы цикла WHILE. Сначала задаём числовую переменную sum равную 0. Затем начинаем цикл WHILE с условием выполнятся до тех пор, пока переменная sum ни будет равна 100. Затем в теле цикла мы прибавляем к переменной sum единичку, тем самым задаётся шаг цикла равный одному. И с каждым проходом цикла переменная sum будет увеличивается на 1 до тех пор, пока не будет равна 100.

Рассмотрим конструкцию цикла WHILE. Сперва указывается начало цикла командой WHILE затем идёт условие (в примере sum<>100), затем идёт команда DO указывающая на начала тела цикла. Затем записывается само тело цикла и в конце цикла потщится команда ENDWHILE.

6. МАССИВЫ

Массив – это структура данных, которая содержит в себе набор компонентов (элементов массива), которые расположены в памяти непосредственно друг за другом. Каждый элемент массива индексируется (указывается ссылка на номер позиции элемента массива).

Пример массива.

```
VAR string arr{3} := ["first","second","last"];
```

В этом примере дан массив, состоящий из 3-ёх элементов. Чтобы указать, что переменная является массивом нужно после названия переменной добавить фигурные скобки {} и в них указать размерность массива. Элементы массива записываются в квадратных скобках [] через запятую. Индексирование элементов начинается с 1 до длинен массива. Чтобы получить элемент массива нужно указать название массива и в фигурных скобках индекс элемента. Допустим нужно из массива вывести получить слово «second», тогда нам нужно посмотреть на каком месте в массиве находится это слово. Счёт начинается с лева на право, и получается, что нужное нам слово «second» находится на 2 месте и имеет индекс 2.

```
VAR string second:= arr{2};
```

7. ИСТОЧНИКИ

1) Variables [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc6.html (17.02.2017)
2) Persistent variables [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc7.html (17.02.2017)
3) Constants [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc8.html (17.02.2017)
4) Operators [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc9.html (17.02.2017)

5) IF/ELSE [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc10.html (17.02.2017)
6) FOR loop [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc12.html (27.02.2017)
7) WHILE loop [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc13.html (27.02.2017)
8) Arrays [WWW] http://developercenter.robotstudio.com/BlobProxy/manuals/IntroductionRAPIDProgOpManual/doc26.html (27.02.2017)