# TRURL

Reverse Polish Notation for Object Pascal

***RPN Engine***

---

Lazarus and Free Pascal Edition

Version 1.0
Document version 1.0.1
2019-07-13

Johannes W. Dietrich

---

The TRURL RPN engine is a library of Object Pascal types, classes and procedures that can be used as a foundation for virtual calculators using Reverse Polish Notation (RPN).

The unit “rpnengine.pas” is the basis of the library. It is necessary for all projects using the RPN engine and sufficient for both command-line and text-mode user interfaces. For applications based on a graphical user interface (GUI), the optional unit “rpnwidgets.pas” provides additional functionality, which facilitates linking to pre-defined interface elements.

## About Reverse Polish Notation

*Reverse Polish Notation*, abbreviated *RPN*, also known as *Polish postfix notation* or *postfix notation* is a mathematical notation, in which operators follow their operands. It goes back to the older *Polish Notation*, which was suggested in 1924 by the philosopher, mathematician and logician Jan Łukasiewicz.

For programming computers and calculators RPN has many advantages over the more common algebraic notation, since it makes parenthesis unnecessary and saves time and key strokes. Additionally, it facilitates detection and correction of errors.

These reasons motivated producers of advanced scientific calculators including Hewlett-Packard (HP), Elektronika, Semico, SwissMicros and others to base their models on RPN logic.

RPN is also used by some well-known software packages including dc, xcalc, RPL and Calc in Emacs, and by stack-oriented programming languages, e.g. Forth and PostScript.

More information on RPN is available from:

https://www.swissmicros.com/what_is_rpn.php

https://www.calculator.org/articles/Reverse_Polish_Notation.html

http://www.hp-prime.de/files/composite_file/file/192-hp_rpn_en.pdf

https://www.hpmuseum.org/rpn.htm

## Unit rpnengine.pas

The basic unit rpnengine.pas is required for every project, irrespectively whether or not it uses a graphical user interface (GUI). It provides the following constants, types, data structures, classes and functions:

### Basic types

```
TBinOperator = (PlusOp, MinusOp, MultOp, DivOp, PowerOp);

TUniOperator = (PlusMinusOp, InvertOp, SinOp, CosOp, TanOp,
ASinOp, ACosOp, ATanOp, sqrtOp);

TAngleMode = (Degree, Radian, Turn, Grad);
```

These global types are necessary for the engine, but also usable by your code. *TUniOperator* and *TBinOperator* represent unary and binary operators, respectively, which are used by *TEngine.rpn*. *TAngleMode* determines the mode, how angles are encoded.

### Global constants

Version information:

```
  RPNEngine_major   = 1;
  RPNEngine_minor   = 0;
  RPNEngine_release = 0;
  RPNEngine_patch   = 0;
  RPNEngine_fullversion = ((RPNEngine_major * 100 +
RPNEngine_minor) *
    100 + RPNEngine_release) * 100 + RPNEngine_patch;
  RPNEngine_version = '1.0.0.0;
  RPNEngine_internalversion = 'Aleph';
```

## TStack

```
TStack = class
private
  {private fields}
protected
  {protected fields and methods}
public
  constructor create;
  destructor destroy; override;
  procedure Clear;
  procedure RollDown;
  procedure DropDown;
  procedure RollUp;
  procedure Push(operand: extended);
  function Pop: extended;
  procedure Error(msg: String);
public
  property x: extended read fx write fx;
  property y: extended read fy;
  property z: extended read fz;
  property t: extended read ft;
  property lastx: extended read fl;
end;
```

*TStack* implements a virtual stack with four registers (*x*, *y*, *z* and *t*) and a special *lastx* register, which holds the content of the *x* register as it was before execution of certain operations.

| | | |
|---|---|---|
| T | **7.89** | |
| Z | **4.56** | |
| Y | **3.14** | |
| X | **1.23** | (Shown in the display by most calculators) |

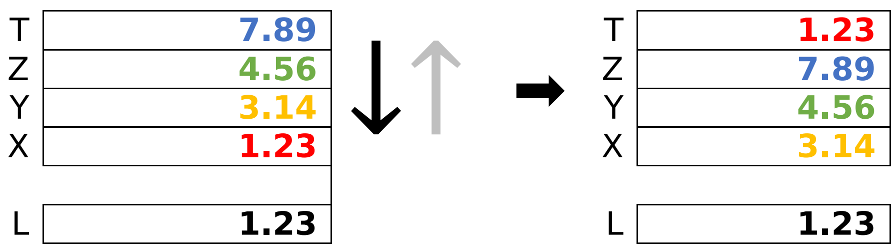
| | |
|---|---|
| L | **1.23** |

*TStack.Clear* clears the virtual stack. This method sets all registers to 0.

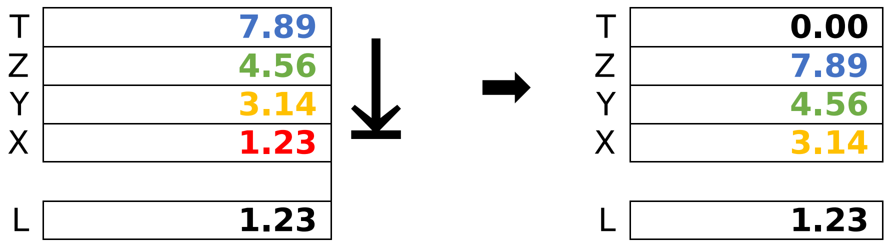
| | |
|---|---|
| T | **0.00** |
| Z | **0.00** |
| Y | **0.00** |
| X | **0.00** |

| | |
|---|---|
| L | **0.00** |

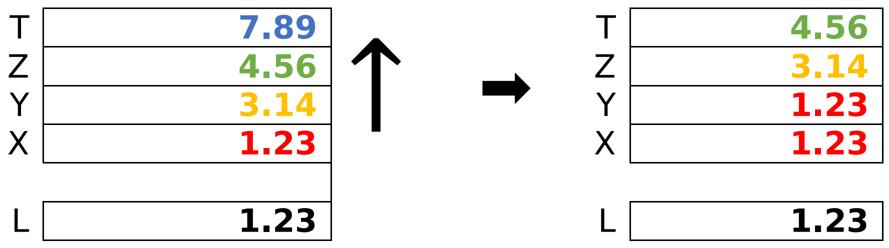
*TStack.Rolldown* rolls the registers in the virtual stack down and recycles the contents of the *x* register in the *t* register.



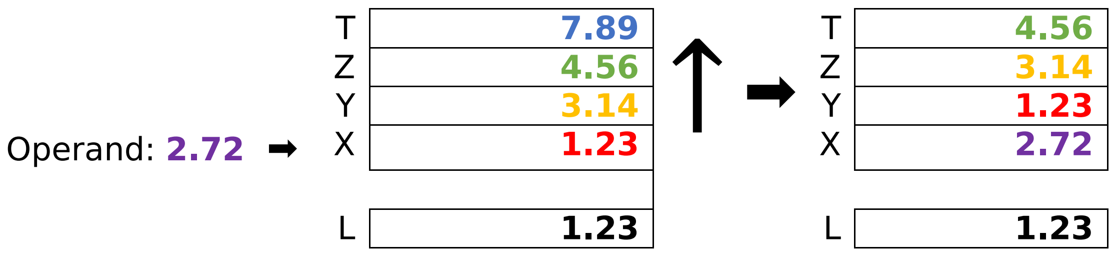
*TStack.DropDown* drops the registers in the virtual stack down. It doesn’t recycle contents, so that the *t* register receives the value 0.



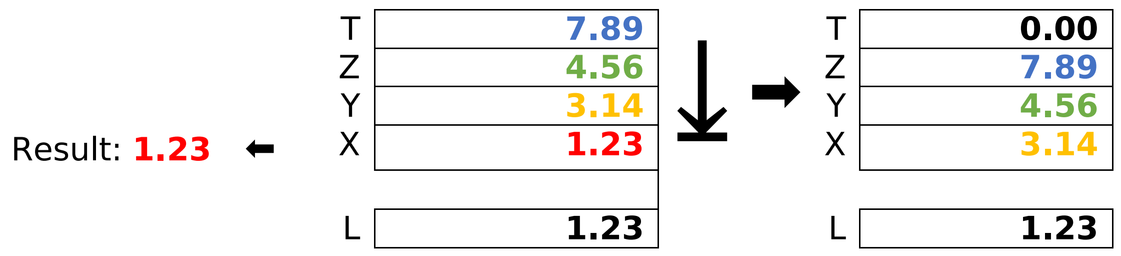
*TStack.Rollup* rolls the registers in the virtual stack up. This method copies the contents of the *x* register to the *y* register.



*TStack.Push* pushes the content of an operand to the stack, i.e. the *x* register gets the value of the operand and the stack is lifted.



*TStack.Pop* delivers the contents of the *x* register and drops down the stack.



*TStack.Error* is predominantly employed by the engine itself for reporting error conditions, but also usable by your code.

## TEngine

```
TEngine = class
public
  Stack: TStack;
  AngleMode: TAngleMode;
  constructor create;
  destructor destroy; override;
  procedure Add;
  procedure Sub;
  procedure Times;
  procedure Divide;
  procedure CHS;
  procedure Inv;
  procedure PWR;
  procedure Sinus;
  procedure Cosinus;
  procedure Tangens;
  procedure ArcSinus;
  procedure ArcCosinus;
  procedure ArcTangens;
  procedure sqroot;
  function rpn(operand1, operand2: extended; binOp:
TBinOperator): extended;
  function rpn(operand: extended; uniOp: TUniOperator):
extended;
end;
```

*TEngine* is a class which implements a virtual calculation engine. It provides the following variables and methods:

*TEngine.Stack* refers to the engine's stack object (of type *TStack*). It is automatically created by the *create* method of *TEngine* and cleared, if the engine object is *destroyed*.

*TEngine.AngleMode* is a global variable, which holds the mode of encoding angles. Its default value is *Degree*.

*TEngine.Add* adds the two numbers in the *x* and *y* registers and stores the result in the *x* register. The *lastx* register is supported.

*TEngine.Sub* subtracts the two numbers in the *y* and *x* registers and stores the result in the *x* register. The *lastx* register is supported.

*TEngine.Times* multiplies the two numbers in the *x* and *y* registers and stores the result in the *x* register. The *lastx* register is supported.

*TEngine.Divide* divides the two numbers in the *y* and *x* registers and stores the result in the *x* register. The *lastx* register is supported.

*TEngine.CHS* toggles the sign of the number in the *x* register.

*TEngine.Inv* inverts the number in the *x* register. The *lastx* register is supported.

*TEngine.PWR* raises the number in the *y* register to that in the *x* register and stores the result in the *x* register.

*TEngine.Sinus* calculates the sine of the number in the *x* register. The result depends on the value in the *TEngine.AngleMode* variable. The *lastx* register is supported.

*TEngine.Cosinus* calculates the cosine of the number in the *x* register. The result depends on the value in the *TEngine.AngleMode* variable. The *lastx* register is supported.

*TEngine.Tangens* calculates the tangent value of the number in the *x* register. The result depends on the value in the *TEngine.AngleMode* variable. The *lastx* register is supported.

*TEngine.ArcSinus* calculates the inverse sine of the number in the *x* register. The result depends on the value in the *TEngine.AngleMode* variable. The *lastx* register is supported.

*TEngine.ArcCosinus* calculates the inverse cosine of the number in the *x* register. The result depends on the value in the *TEngine.AngleMode* variable. The *lastx* register is supported.

*TEngine.ArcTangens* calculates the inverse tangent value of the number in the *x* register. The result depends on the value in the *TEngine.AngleMode* variable. The *lastx* register is supported.

*TEngine.sqroot* calculates the square root of the number in the *x* register. The *lastx* register is supported.

The function *TEngine.rpn* is a polymorphic function, which accepts operands and operators, and calculates the result, which is returned as floating point number of type extended. The two variants

*TEngine.rpn(operand1, operand2: extended; binOp: TBinOperator)*
and
*TEngine.rpn(operand: extended; uniOp: TUniOperator)*

use the following binary and unary operators:

*PlusOp:* returns the sum of *operand1* and *operand2*.

*MinusOp:* calculates the difference of *operand1* and *operand2*.

*MultOp:* returns the product of *operand1* and *operand2*.

*DivOp:* calculates the ratio of *operand1* and *operand2*.

*PowerOp:* returns *operand1* raised to the power of *operand2*.

*PlusMinusOp:* toggles the sign of *operand*.

*InvertOp:* inverts *operand*.

*SinOp:* returns the sin *of operand.*

*CosOp:* calculates the cosine *of operand.*

*TanOp:* delivers the tangent *of operand.*

*ASinOp:* returns the inverse sine *of operand.*

*ACosOp:* calculates the inverse cosine *of operand.*

*ATanOp:* delivers the inverse tangent *of operand.*

*sqrtOp:* calculates the square root *of operand.*

## Unit rnpwidgets.pas

The unit rpnwidgets.pas helps to create a GUI-based virtual calculator:

### Types

```
TEntryMode = (PostOper, PostEnter, Number);
```

### TFrame

```
TFrame = class
private
  {private fields}
public
  Engine: TEngine;
  TRegDisplay, ZRegDisplay, YRegDisplay, XRegDisplay:
TControl;
  EntryMode: TEntryMode;
  constructor create;
  destructor destroy; override;
  procedure HandleEnter;
  procedure HandleClear;
  procedure HandleInv;
  procedure HandleAdd;
  procedure HandleSub;
  procedure HandleTimes;
  procedure HandleDiv;
  procedure HandleCHS;
  procedure HandlePWR;
  procedure HandleSin;
  procedure HandleCos;
  procedure HandleTan;
  procedure HandleASin;
  procedure HandleACos;
  procedure HandleATan;
  procedure HandleSqrt;
  procedure HandleRollDown;
  procedure DisplayRegisters;
  procedure AppendChar(ch: char);
  procedure InsertString(theString: String);
  procedure CheckEngine;
  procedure Error(msg: String);
end;
```

TFrame provides an object-oriented approach for creating a GUI-based calculator with the following objects and methods:

*TFrame.Engine* links to an engine object of type *TEngine*. The engine object is not automatically created with *TFrame*'s create method and not automatically disposed of on *TFrame.destroy*. It is the responsibility of the programmer to create and clear this object, where necessary.

*TFrame.TRegDisplay*,
*TFrame.ZRegDisplay*,
*TFrame.YRegDisplay*,
and *TFrame.XRegDisplay* point to an object of type *TControl*. Assigning them to a control of your project (e.g. written with the Lazarus IDE) ensures automatic output to a virtual display provided by you. The RPN Widgets use the *caption* property of this control.

*TFrame.EntryMode*: This selector is mainly for internal purposes of the engine, but it may be used by custom programs, too.

*TFrame.Error* is predominantly employed by the engine itself for reporting error conditions, but also usable by custom code.

*TFrame.HandleEnter:* Handles pressing the “Enter” key by rolling the stack up. If assigned, the display is updated automatically.

*TFrame.HandleClear:* Clears the *x* register. If assigned, the display is updated automatically.

*TFrame.HandleInv:* Inverts the contents of the *x* register and rolls the stack up. If assigned, the display is updated automatically.

*TFrame.HandleAdd*, *TFrame.HandleSub*, *TFrame.HandleTimes*, *TFrame.HandleDiv*, *TFrame.HandleCHS*, *TFrame.HandlePWR*, *TFrame.HandleSin*, *TFrame.HandleCos*, *TFrame.HandleTan*, *TFrame.HandleASin*, *TFrame.HandleACos*, *TFrame.HandleATan* and *TFrame.HandleSqrt* trigger the corresponding actions of the engine and update the entry mode. If assigned, the display is updated automatically, too.

*TFrame.HandleRollDown*: Rolls down the stack and updates the display, if assigned.

*TFrame.DisplayRegisters*: Displays the register contents in a control, which is provided by the GUI of the app. This procedure is called by most operations automatically.

*TFrame.AppendChar(ch: char):* Depending on *TFrame.EntryMode* this procedure inserts a digit in the *x* register, rolls the stack or toggles the entry mode. If assigned, the display is updated automatically.

*TFrame.InsertString(theString: String):* inserts a string representing a floating-point number into the *x* register. If assigned, the display is updated automatically.

## Examples

The program TRURL A, available from http://trurl.sf.net, provides an example for a simple calculator based on the RPN Engine. Both source code and pre-complied sample implementations for macOS, Windows and other operating systems are provided.

## Contact

PD Dr. med. Johannes W. Dietrich, Laboratory XU44, Medical Hospital I, Bergmannsheil University Hospitals, Ruhr University of Bochum, Bürkle-de-la-Camp-Platz 1, D-44789 Bochum, NRW, Germany

http://**trurl**.sf.net