MODULE 13: MATHEMATICS

On completion of this module you will be able to make use of mathematical formulas to do calculations for your program to give accurate and fast outputs for the user.

MODULE 13.1: COMMANDS

Subject Outcome 1: Logical Operations

Subject Outcome 2: String & Numerical mathematical functions

Subject Outcome 3: Conditional mathematical functions

Subject Outcome 4: Advanced Mathematical functions

Subject Outcome 5: Circular mathematical formula

Subject Outcome 6: ABS, Cint, Fix, Frac, Ceil, Floor, Round functions

Subject Outcome 7: Sgn

Subject Outcome 8: STRF$() format

Subject Outcome 9: Distance between two points

Subject Outcome 10: Determine Angle between two points

13.1 LOGICAL OPERATIONS

Operators perform mathematical or logical operations on values. They are usually encompassed by an expression for example; 2 * 8 is a valid expression; and * is an operator operating on values 2 and 8 making it 16.

The highest precedence operator will always execute before any lower precedence operator. For operators that share the same precedence, the left to right associativity law holds (2+5-1+27). The priority is:

- Brackets: 2+(5-1)+27
- Division (/)
- Multiply (*)
- Subtraction (-)
- Addition (+)

Mathematical procedures are divided into four (4) main groups:

- Simple (7+8) – remember you need STR$() to display the result.
- Advance (sin, cos, tan)
- Formulas (speed=d/v)
- String ("hallo - lo"+"andre")

Here is a list of valid arithmetic operators (the ? represents any given number):

OPERATOR	COMMAND	CODE
String Index – select specific entry on the string.	S$(?)	$TYPECHECK ON dim s$ as string s$="Mil" showmessage s$[2]
Exponentiation – calculates power of a number	?^?	**Showmessage str$(2^6)**
Negation - a negative number	-?	**Showmessage str$(7-14)**
Multiplication – multiplies 2 numbers	?*?	**Showmessage str$(2*6)**
Floating point-division – divide 2 floating point numbers (include the .? as part of the result)	/	**Showmessage str$(2.5/2.6)**
Integer Division – divide 2 integer numbers. Before the division is done, the numbers are rounded off.	6\2	**Showmessage str$(6.4\2.3)**

OPERATOR	COMMAND	CODE
Left Bit Shift – shifts bit by amount specified for instance 10 shl 2 = (10 * 2)*2	? shl ?	Showmessage str$(10 shl 2)
Right Bit Shift – shifts bit by amount specified to the right for instance (10/2)/2	? shr ?	Showmessage str$(10 shr 2)
Modulus/remainder – this will display the remainder of a division.	? mod ?	Showmessage str$(15 mod 10)
Inverse Modulus - returns the inverse of a number in modulus (how many times to be divided + 1)	? inv ?	Showmessage str$(3 inv 26)
Addition – adding numbers or strings.	?+?	Showmessage str$(3+6) Showmessage "Hi "+"Andre"
Subtraction - Subtracts 2 operands (number or strings)	? - ?	Showmessage str$(10-4) Showmessage "Jello"-"jeo"
Grouping to set priority – using brackets to set the priority calculations	(?...?)	Showmessge str$(6+(3*2)+4)
SQR - square root	SQR(?)	Showmessage str$(sqr(9))

13.2 STRING & NUMERIAL VARIABLES FUNCTIONS

As seen in the previous module, you may add and subtract strings just as you would do numbers. Remember the following however:

- Use the STR$() command to display a numerical variable.
- Use the VAL() command to covert a string as a numerical variable.

13.3 CONDITIONAL MATHEMATICAL FUNCTIONS

Relation operators (conditional) are used to compare 2 values. The result of this comparison is either true or false. These operators are used with the **IF ... THEN command**.

OPERATOR	COMMAND	CODE
Equality - test for equality between 2 operands (strings and numbers)	=	$TYPECHECK ON dim a as single:a=5 dim b as single:b=5 dim jk as string:jk="andre" dim kk as string:kk="johan" if a=b then showmessage "yes number" end if if jk=kk then showmessage "yes person" end if

OPERATOR	COMMAND	CODE
InEquality - test for non-equality between 2 operands (strings and numbers)	<>	$TYPECHECK ON dim a as single:a=5 dim b as single:b=5 dim jk as string:jk="andre" dim kk as string:kk="johan" if a<>b then showmessage "yes number" end if if jk<>kk then showmessage "yes person" end if
Less Than - test if the operand is less than. **<= is for less and equal**. Remember something = will always be missed if you only check for < or > as it is in between.	< <=	$TYPECHECK ON dim a as single:a=5 **dim b as single:b=7** dim jk as string:jk="andre" dim kk as string:kk="johan" **if a<=b then** **showmessage "yes number"** **end if**
Great Than - test if the operand is greater than. **>= is for greater and equal.** Remember something = will always be missed if you only check for < or > as it is in between.	> >=	$TYPECHECK ON **dim a as single:a=8** dim b as single:b=5 dim jk as string:jk="andre" dim kk as string:kk="johan" **if a>=b then** **showmessage "yes number"** **end if**
Compare if corresponding multiple evaluations to be the same	AND	$TYPECHECK ON dim a as single:a=5 dim b as single:b=5 dim jk as string:jk="andre" dim kk as string:kk="johan" **if a=5 and b=5 then** **showmessage "yes number"** **end if**
Compare with operands not being the same or if one of the two is true/valid.	OR	$TYPECHECK ON dim a as single:a=5 dim b as single:b=5 dim jk as string:jk="andre" dim kk as string:kk="johan" if a=5 and b=5 then showmessage "yes number" end if **if jk or kk ="johan" then** **showmessage "yes person"** **end if**

13.4 ADVANCED MATHEMATICS (scientific functions)

I will not dwell too much on these functions as they are mostly used for advanced graphics, scientific calculations and game programming. Unless you code any of these mentioned programs, you will never use it. I will however mention them should you wish to code them for your program. All these elements holding values related to these calculations must be **DIM** as **DOUBLE**. The ? in the samples represents any given number.

OPERATOR	COMMAND
ACOS – function that returns the arccosine of a numeric expression	ACOS(?)
ASIN – function that returns the arcsine of a numeric expression	ASIN(?)
ATN/ATAN - numeric expression is the angle expressed in radians	ATN(?)
COS - numeric expression is the angle expressed in radians	PI=3.14153 COS(pi)
LOG – function that returns the natural logarithm of a numeric expression	LOG(10)
SIN – numeric expression is the angle expressed in radians	PI=3.14154 SIN(pi)
TAN - numeric expression is the angle expressed in radians	TAN(?)
EXP – a math function that returns the exponential function (raised to the power of n - … =	EXP(?)
SQR – function that returns the square root of a number	SQR(?)

13.5 CIRCULAR DETECTION

I will mention this example as it is used in a variety of programs and it is a good example on how to use advanced mathematics. The idea is to use maths to determine if the user has clicked inside or outside a circle (special buttons, etc.)

CODING	EXPLANATION
$TYPECHECK ON	
$INCLUDE "rapidq.inc"	
DECLARE SUB HitCircleEvent(d AS SINGLE)	Event when clicking
TYPE QCircle EXTENDS QCanvas	Create an ELEMENT QCIRCLE
OnHitCircle AS EVENT(HitCircleEvent)	
	Activate the event to refresh the
EVENT OnPaint	screen.
WITH QCircle	
.Circle(0, 0, .Width, .Height, clWindowText, clBtnFace)	
END WITH	
END EVENT	
EVENT OnMouseDown(Button AS LONG, x AS LONG, y AS LONG, Shift AS LONG)	Activate the mouse button click
WITH QCircle	event.
dim d as single	
d = SQR((x - .Width / 2) ^ 2 + (y - .Height / 2) ^ 2)	**Calculation to determine position**
IF d <= .Width / 2 THEN CALLFUNC(.OnHitCircle, d)	**and distance from center of the**
END WITH	**circle** – this will then determine if
END EVENT	the distance is further than the
CONSTRUCTOR	radius of the circle, if less then
Width = 100:Height = 100	you clicked in the circle.
END CONSTRUCTOR	
END TYPE	
	If a hit is registered, then display
SUB Circle1_HitCircle(d AS SINGLE)	message.
SHOWMESSAGE "Distance from center: " + STR$(d)	
END SUB	
	Window/Form.
CREATE Form AS QForm	
Width = 300	
Height = 300	Establish element QCIRCLE
CREATE Circle1 AS QCircle	Start the event mouse click
OnHitCircle = Circle1_HitCircle	handler.
END CREATE	
Center	
END CREATE	
form.showmodal	

13.6 MANIPULATION FUNCTIONS – ABS, CINT, FIX, FRAC

13.6.1 ABS

This math function will return the absolute value of a numeric expression. In other words, it will turn any negative (-) number and make it a positive expression.

OPERATOR	COMMAND	CODE
ABS	ABS(-?)	$TYPECHECK ON dim a as single:a=-5 dim b as single b=abs(a) showmessage str$(b)

13.6.2 CINT

This conversion function converts a numeric expression (SINGLE, DOUBLE) to an INTEGER by rounding the fractional part of the expression.

OPERATOR	COMMAND	CODE
CINT	Cint(?)	$TYPECHECK ON dim a as single:a=41.2 dim b as single b=cint(a) showmessage str$(b)

13.6.3 FIX

A function that removes the fractional part of a number (the fraction is beyond ?.?? – the ??)

OPERATOR	COMMAND	CODE
FIX	Fix(?)	$TYPECHECK ON dim a as single:a=41.2 dim b as single b=fix(a) showmessage str$(b)

13.6.4 CEIL

This math function rounds a numeric expression up towards positive infinity (next number, regardless if it is less than .5)

OPERATOR	COMMAND	CODE
CEIL	Ceil(?)	$TYPECHECK ON dim a as single:a=41.2 dim b as single b=ceil(a) showmessage str$(b)

13.6.5 FRAC

This function returns the fractional part of a number. Remember always to work with DOUBLE when working with numbers with fractions.

OPERATOR	COMMAND	CODE
FRAC	Frac(?)	$TYPECHECK ON dim a as double:a=41.2 dim b as double b=frac(a) showmessage str$(b)

13.6.6 FLOOR

A math function that rounds a numeric expression down towards negative infinity. This means that it will always return the lower value (*50.4 = 50*, *50.8 = 50*).

OPERATOR	COMMAND	CODE
FLOOR	Floor(?)	$TYPECHECK ON dim a as double:a=41.7 dim b as double b=floor(a) showmessage str$(b)

13.6.7 ROUND

This function will convert a number to an integer by rounding the fractional part of the expression (if it is less than .5 then lower value, if .5 and more then higher value).

OPERATOR	COMMAND	CODE
ROUND	Round(?)	$TYPECHECK ON dim a as double:a=41.7 dim b as double b=round(a) showmessage str$(b)

13.7 SGN

This function indicates the sign of numeric expression (+ - or 0). In other words it will indicate whether a number is a negative, positive or a 0 (zero) number. If the SGN result is (1) then it is a positive value, if (0) then it is a zero and if (-1) then the number is a negative. This is **useful as all programs will crash if you divide by 0 or by a negative number**, so first **test before** your **calculate** with a **division**.

OPERATOR	COMMAND	CODE
SGN	SGN(?)	$TYPECHECK ON dim a as double:a=-41.7 dim b as double b=sgn(a) showmessage str$(b)

13.8 STRF$(...,?,??,????)

This is a conversion function that returns a formatted string representation of the value of a numeric expression. The ... represents the **number**, ? represents the **FORMAT**, ?? represents the **PRECISION** and ??? represents the **DIGITS**. This is mostly used with funds (money R200.10) related numbers that has to be rounded with only two fractions (.10).

The **FORMAT** consist out of one of the following:

- FFGENERAL: converts to the shortest possible decimal string when trailing zeros are removed (mostly used).
- FFEXPONENT: converts to a scientific notation of the form.
- FFFIXED: converts to fixed point format of the form.
- FFNUMBER: converts to a number format of the form.

PRECISION% specifies how many decimal places to calculate (total characters). **What is important** is to take note that this function is a **STRING** and not a **NUMBER**. You need the VAL to convert it as a number.

OPERATOR	COMMAND	CODE
STRF$	STRF$(...,?,??,????)	$TYPECHECK ON $include "rapidq2.inc" dim a as string dim b as double **a=strf$(150.3563,ffgeneral,3,0)** b=val(a) showmessage str$(b)

13.9 DISTANCE BETWEEN TWO LINES

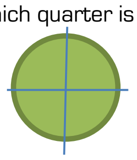
When you have two points on map or image, you need to determine the distance. Here is the formula to do just that. We will code a CANVAS with a red circle in the middle. Move the mouse around to see the distance from the red circle.

CODING	EXPLANATION
$TYPECHECK ON	
$INCLUDE <RapidQ2.inc>	
declare sub mousepos	
declare sub drawme	
dim snake1 as double:snake1=303	The X position of the circle (from point position)
dim snake2 as double:snake2=203	The Y position of the circle (from point position)
dim snake3 as double	The position of the mouse cursor will be recorded here (X)
dim snake4 as double	The position of the mouse cursor will be recorded here (Y)
dim a$ as string	
CREATE Form AS QFORMex	
Caption = "Form":Width = 640:Height = 480:Center	
create canvas as qcanvas	
width=600:height=400	
onmousemove=mousepos	Start event trapper for mouse movement.
onpaint=drawme	
end create	

CODING	EXPLANATION
END CREATE	
SetWindowLong(Form.Handle, -8, 0)	
SetWindowLong(Application.Handle, -8, Form.Handle)	
Form.ShowModal	
sub mousepos	Assign the value of the mouse X position and mouse Y
snake3=mousex:snake4=mousey	position.
A$ = str$(sqr(((snake3-snake1)^2)+((snake4-snake2)^2)))	The formula to determine the distance (remember the
form.caption="Distance: "+a$	position of **snake1** and **snake2** are fixed.
end sub	
sub drawme	
canvas.circle(300,200,306,206,255,255)	Draw the circle in the middle of the canvas.
end sub	

13.10 ANGLES

To determine the angel between two points (in mills – 0 to 6400 mills, degrees work from 0 – 360 however it is not as accurate. Should you however wish to work with degrees, then divide the result with 17.777778.

CODING	EXPLANATION
$TYPECHECK ON	
$INCLUDE <RapidQ2.inc>	
declare sub mousepos	
declare sub drawme	
dim x1 as double:dim x2 as double	Create variables that will be used to calculate
dim x3 as double:dim y1 as double	the angle.
dim y2 as double:dim y3 as double	
dim angle as double	
CREATE Form AS QFORMex	
Caption = "Form":Width = 640:Height = 480:Center	
create canvas as qcanvas	
width=600:height=400	
onmousemove=mousepos	As the mouse cursor moves, it will determine
onpaint=drawme	the angle from the middle point (red circle)
end create	
END CREATE	
SetWindowLong(Form.Handle, -8, 0)	
SetWindowLong(Application.Handle, -8, Form.Handle)	
Form.ShowModal	
sub mousepos	The **FROM** point is fixed (middle red circle)
x1=303:y1=203	The **TILL** position is the position of the mouse.
x2=mousex:y2=mousey	Calculate the Z position.
x3=x2-x1:y3=y2-y1	Determine in which quarter is the line.
if x3=0 and y3<0 then angle=6400:goto skipallzz	
if x3>0 and y3=0 then angle=1600:goto skipallzz	
if x3=0 and y3>0 then angle=3200:goto skipallzz	
if x3<0 and y3=0 then angle=4800:goto skipallzz	
if y3=0 then goto skipallzz	
if x3=0 then goto skipallzz	
angle=((atan(y3/x3)*180/3.14153))*17.777778	Calculate the angle (in mills) – take out the
if x1<x2 and y1>y2 then angle=1600+angle	*17.777778 to determine degrees.
if x1<x2 and y1<y2 then angle=1600+angle	Place within the quarter.
if x1>x2 and y1<y2 then angle=3200+(angle+1600)	
if x1>x2 and y1>y2 then angle=4800+angle	
skipallzz:	

CODING	EXPLANATION

form.caption=str$(angle)+" mills"+" "+str$(angle/17.777778)+"º"
canvas.fillrect(0,0,600,400,16777215)
canvas.line(x1,y1,x2,y2,255)
end sub
 	
sub drawme
   canvas.fillrect(0,0,600,400,16777215)
   canvas.circle(300,200,306,206,255,255)
end sub
	Display result (mills) and degrees.
	Clear the canvas with a filled rectangle.
	Draw a line to see the angle.