MODULE 9: FLOW OF THE PROGRAM : BRANCHING

On completion of this module you will be able to teach the program how to scan through data and move within the program to the option as required.

Subject Outcome 1: Introduction (Branching)

Subject Outcome 2: Calling a sub-program within a sub-program

Subject Outcome 3: Goto & Labels

Subject Outcome 4: Gosub & Return

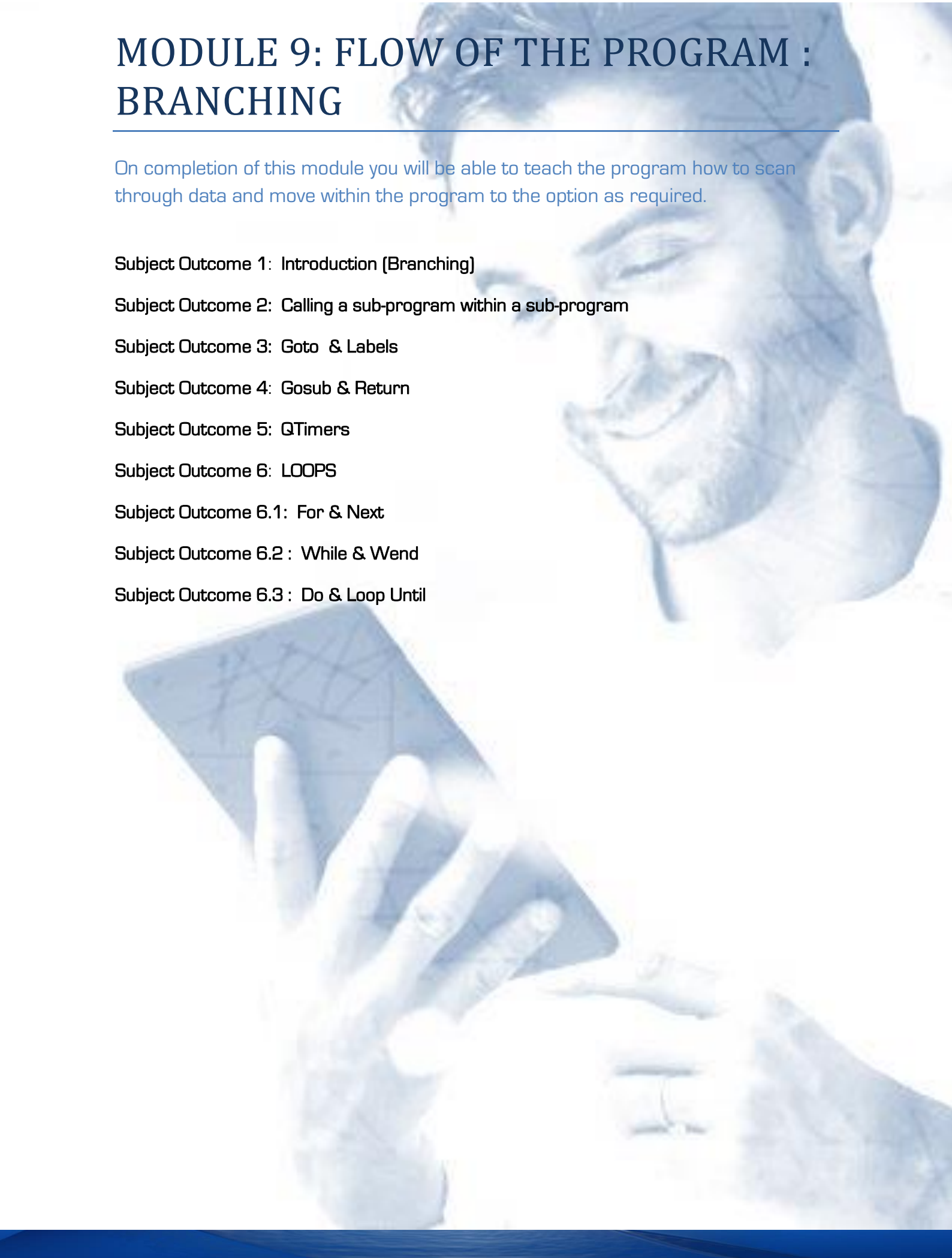
Subject Outcome 5: QTimers

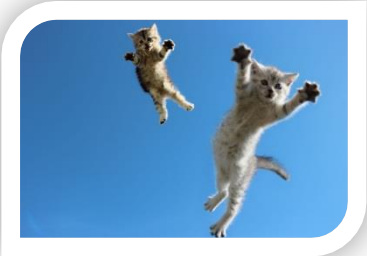
Subject Outcome 6: LOOPS

Subject Outcome 6.1: For & Next

Subject Outcome 6.2 : While & Wend

Subject Outcome 6.3 : Do & Loop Until

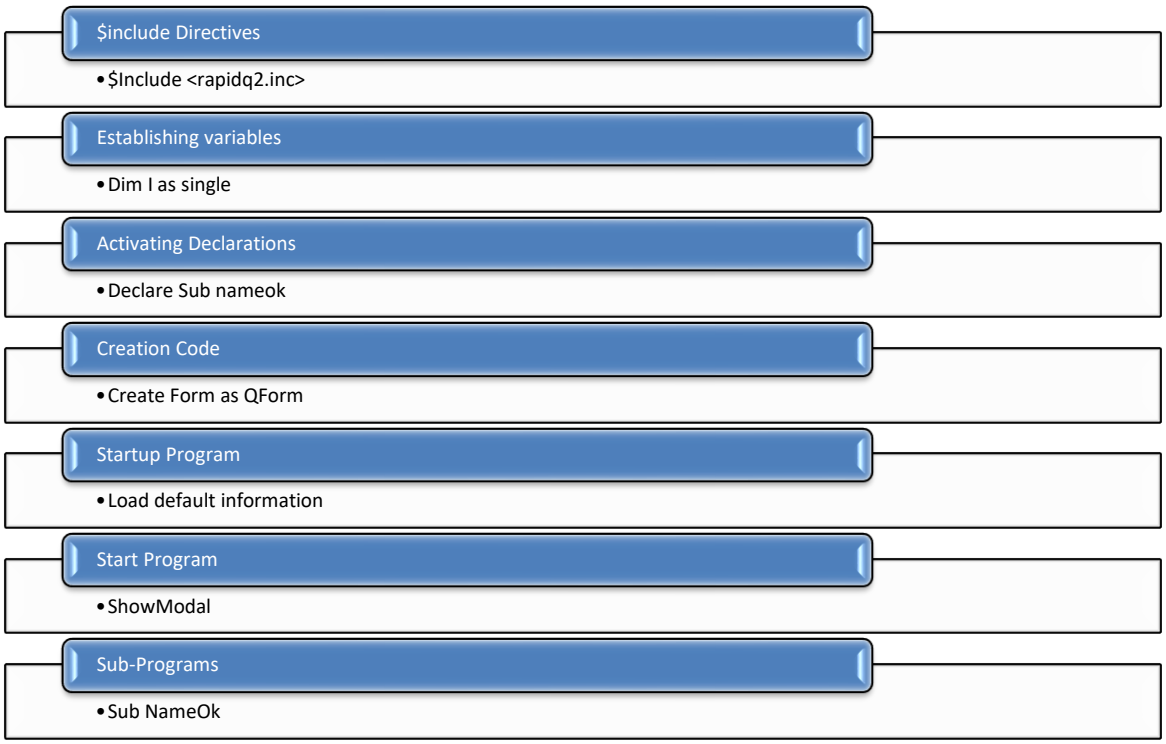




9.1 FLOW OF THE PROGRAM

9.1.1 Introduction

Branching is the mechanism used of skipping (jumping) instructions (coding lines), either **forward** or **backwards**.
Branching can be seen as sub-sub routines **within** your main program or **sub-programs**.
To understand branching it is important that you be reminded again on the **structure** of a program:



A sub-program is called by activating it (declare sub) and then by calling it by its name to be used. Branching is done the **same way, except** you **don't activate** it, it is within the sub-programs. There are multiple methods of calling a branched routine:

9.1.2 Calling a Sub-Program within a Sub-Program

A Sub-Program is not just called bmo ONCLICK, ONMOUSEDOWN, ONKEYUP etc. You may call it within the sub-program itself as well. There are however two methods of doing this.

Syntax (with no parameters):

Commands	Description
$TYPECHECK ON $INCLUDE <RapidQ2.inc> **declare sub callthisoption** **declare sub callthatopt** CREATE Form AS QFORM Caption = "Click on Window":Width = 640 Height = 480:center **onclick=callthisoption** END CREATE SetWindowLong(Form.Handle, -8, 0) SetWindowLong(Application.Handle, -8, Form.Handle) Form.ShowModal **sub callthisoption** **form.circle(50,50,100,100,255,-1)** callthatopt **end sub** **sub callthatopt** **showmessage "hallo"** **end sub**	There are no extensions or special variable holders when calling the sub-program. Declare sub program. Declare sub program. When you click on the window then call sub program CALLTHISOPTION. Sub program CALLTHISOPTION Call sub program CALLTHATOPT

As you can see above, you may call another sub-program from within a sub program.

Syntax (with parameters):

Commands	Description
$TYPECHECK ON $INCLUDE <RapidQ2.inc> **declare sub callthisoption** **declare sub** callthatopt(x%,y%) CREATE Form AS QFORM Caption = "Click on Window":Width = 640 Height = 480:center onclick=callthisoption END CREATE SetWindowLong(Form.Handle, -8, 0) SetWindowLong(Application.Handle, -8, Form.Handle) Form.ShowModal sub callthisoption form.circle(50,50,100,100,255,-1) callthatopt(screen.mousex,screen.mousey) end sub **sub callthatopt(x%,y%)** **showmessage "mouse pos: "+str$(x%)+"/"+str$(y%)** **end sub**	 We added some parameters that will hold variables **x%** and **y%** (numerical variables since the **% indicates that it is numerical** – **if it was string then the %** would be **$**). As seen here we are using the mouse screen's position to indicate the x% and y% to call the sub-program. The parameters are added that will be filled once the sub-program is called from the caller position. The values x% and y% is being displayed.

Should you have a situation whereby you wish to call the sub-program, but don't care or require the parameters, **you simply call negative or blank parameters**. In the case of a **numerical parameter**: **callthatopt(-1,-1)** and a **string parameter**: **callthatopt("","")**

9.1.3 Goto & Labels

A label identify the starting point of a sub-sub-routine. **You basically create a starting point with a name to be called followed by the : character. This method is used to skip certain coding or simply go to another part** of the sub-program to avoid long processing situations.

Commands	Description

$TYPECHECK ON
$INCLUDE <RapidQ2.inc>
declare sub checkname(key as byte)
CREATE Form AS QFORM
Caption = "Enter Name":Width = 640
Height = 480:center
create edit1 as qedit
left=50:top=50:width=300
onkeydown=checkname
end create
END CREATE
SetWindowLong(Form.Handle, -8, 0)
SetWindowLong(Application.Handle, -8, Form.Handle)
 	
Form.ShowModal
 	
sub checkname(key as byte)
if key=27 then
edit1.text=""
goto endcheckname
end if
if key=13 then
if lcase$(edit1.text)="administrator" then
showmessage "hallo"
end if
end if
endcheckname:
end sub
	Declare the sub program
	
	Check key pressing within the QEDIT field
	
	If the user presses the **ESC** key, clear the **EDIT** field and goto **ENDCHECKNAME** (skip all other coding to avoid slower program).
	
	If the user presses the ENTER key, then check for the entry ADMINISTRATOR.
	
	The label position called **ENDCHECKNAME:**

9.1.4 Gosub & Return

This has the same working procedure as **GOTO**, however it is ended with the command **RETURN**. In other words, **it will go to the label as identified, but return where it last was called at the end of the sub-sub-program.** We use these sub-sub-programs when we want to share same executional commands.

You must however be careful to always place a **GOTO** command just before the starting point of the **Gosub** routine because if there is no return position when the return command is reached, then your program will crash.

Commands	Description

$TYPECHECK ON
$INCLUDE <RapidQ2.inc>
declare sub checkname(key as byte)
CREATE Form AS QFORM
  Caption = "Enter Name":Width = 640
  Height = 480:center
  create edit1 as qedit
  left=50:top=50:width=300
  onkeydown=checkname
  end create
END CREATE
SetWindowLong(Form.Handle, -8, 0)
SetWindowLong(Application.Handle, -8, Form.Handle)
 	
Form.ShowModal
 	
sub checkname(key as byte)
if key=27 then
  gosub endcheckname2
  goto endcheckname
end if
if key=13 then
  if lcase$(edit1.text)="administrator" the
  gosub endcheckname2
  showmessage "hallo"
  end if
end if
goto endcheckname
endcheckname2:
  showmessage "I will clear the field"
  edit1.text=""
return
endcheckname:
end sub
	If **ESC** key is pressed.
	Gosub ENDCHECKNAME2 … execute **showmessagea "I will clear the field":edit1.text=""** … then **return** to the next command directly after the **GOSUB** command, being **goto endcheckname**
	
	Gosub endcheckname2, then execute the commands and return to the next command being **showmessage "hallo"**
	
	Remember then goto command to skip the gosub sub-sub program.
	
	The label for the goto (skip) commands.

As you can see above, with both situations when the **ENDCHECKNAME2** was called using the **Gosub** command, they both wanted the same executional commands – **this is to avoid duplication of coding**.

9.1.5 Timers

When you start coding Artificial Intelligence within your program, you don't want the program to wait for an action provided by the user (**onclick**, **onkey**, **onmousedown**, etc.) The program must out of free will confirm certain activities and actions without having to wait for the user's response. You thereby create timers so that the program at given timings could confirm information without the user's permission/knowledge.

QTIMER implements the Windows API timer function. You may set as many timers as you wish – each timer will have its own function and resulting executional codes when the timer interval is reached. You have 2 types of timers, **QTIMER** and **QDXTIMER**. For standard programs you use **QTIMER**, however for more sophisticated and precise programs such as games and programs recording milliseconds results (athletics), you will

rather use **QDXTIMER** that is a much more refined and precise timer. I will only address **QTIMER** as you are not learning game programming during this course.

QTIMERs are created outside the Source Creation Area. In other words, before you start creating your first QFORM. A Timer is created by a simple **DIM ... AS QTIMER** command.

Sno	Property	Description

1	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	dim tyd1 as qtimer
	CREATE Form AS QFORM
	  Caption = "Check This":Width = 640
	  Height = 480:center
	END CREATE
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
		Create a timer with the name **TYD1.**

2	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	
	declare sub checktyd
	
	dim tyd1 as qtimer:tyd1.ontimer=checktyd
	CREATE Form AS QFORM
	  Caption = "Check This":Width = 640
	  Height = 480:center
	END CREATE
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
	
	sub checktyd
	  form.caption=time$
	end sub
		Declare a sub program **CHECKTYD**.
		
		When the timer's time is reached, then call sub program **CHECKTYD** Due to that this is coded outside the source creation area, you must indicate the object's name whenever you code a property (**TYD1.ONTIMER**).
		
		Sub program **CHECKTYD**.
		Change the Window's heading to indicate the current **system time** (TIME$)

3	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	
	declare sub checktyd
	
	dim tyd1 as qtimer:tyd1.ontimer=checktyd:tyd1.interval=500
	CREATE Form AS QFORM
	  Caption = "Check This":Width = 640
	  Height = 480:center**:color=0**
	END CREATE
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
	
	sub checktyd
	  form.caption=time$
	  form.color=form.color+500
	end sub
		This will set the interval time. Every so many milliseconds, the program must call the declared sub program. **A full second = 1000**, so if you want the program to check every 5 seconds the value will be 5000. **500 = ½ second**. A 100 will be a 10th of a second, etc.
		
		Every ½ second, the color of the window will change to the next value of +500 on the color code.
		
		If you don't set the interval, it will assume every second to execute the sub-program.

Sno	Property	Description
4	$TYPECHECK ON $INCLUDE <RapidQ2.inc> declare sub checktyd **declare sub begintyd** dim tyd1 as qtimer:tyd1.enabled=0 tyd1.ontimer=checktyd:tyd1.interval=500 CREATE Form AS QFORM **Caption = "Click Screen":**Width = 640 Height = 480:center:color=0 **onclick=begintyd** END CREATE SetWindowLong(Form.Handle, -8, 0) SetWindowLong(Application.Handle, -8, Form.Handle) Form.ShowModal sub checktyd form.caption=time$ form.color=form.color+500 end sub sub begintyd tyd1.enabled=1 end sub	The **enabled** property setting will determine if the timer is **active or not.** Some programs will startup loading a lot of information and data into the database. You don't want timers to interfere with this process, so you will immediately after creating the timer disable it (**enabled=0**). You will only activate the timer at the position or area where the timer should start to work. In this situation we added an ONCLICK onto the window screen – if the user click on the screen, the timer will begin (**enabled=1**) In most cases, timers are activated just before the FORM.SHOWMODAL command or maby with a FORM.ONSHOW command. Regardless you are able to control when timers should work or not. Remember each timer must be programmed individually with each sub program. It is also important to turn off the timer if the coding is long to execute as this will create hanging and or duplication of execution for instance: **sub checktyd** tyd1.enabled=0 **form.caption=time$** **form.color=form.color+500** tyd1.enabled=1 **end sub** This will turn the timer off when the timer's sub-program starts and switch it on again when the sub-program is finished.

9.1.6 LOOPS

Loops provide a mechanism for executing a series of instructions a specified number of times. Almost all your programs (especially database related programs) will involve looping. **You have the following loops methods to select from**:

- FOR ... NEXT
- WHILE ... WEND
- DO ... LOOP / LOOP UNTIL

You must however be careful as loops slow your program down. It is crucial that you make use of the best possible mathematical formulas to avoid duplication or multiple conditions before exiting your loops.

9.1.6.1 FOR ... NEXT

The syntax:

```
For i = 1 to 1000
Next  i
```

These commands literally means from number value until value number. What is important to remember is that this is a numerical variable that will hold the position of the loop (i in this situation) – it therefore must be established as a numerical variable (**Dim i as single**).

You may use the values of a string grid to determine from which position till which position for instance (you will convert the string variable as a numerical variable using the **VAL()** command):

```
For i = val(edit1.text) to val(edit2.text)
Next  i
```

You may also count in reverse (10, 9 , 8, 7, 6, etc.) by using the **STEP** command (*example below will count backwards from 1000 to 1 in intervals of 2*):

```
For i = 1000 to 1 step -2
Next i
```

STEP may also be used in forward counting in intervals (*this example will count in intervals of 5*):

```
For i = 1 to 6000 step +5
Next i
```

You may use multiple **FOR** ... **NEXT** commands – this is especially applicable within database searching for information.

```
For i = 1 to 1000 step +2
        For p = 1 to 200 step+1
        Next p
Next i
```

This will however start to slow your program down (large loops).

Commands	Description
$TYPECHECK ON $INCLUDE <RapidQ2.inc> **dim i as single:dim woord as string** **declare sub newaccount** CREATE Form AS QFORM Caption = "Click Screen":Width = 640 Height = 480:center **onclick=newaccount** create lab1 as qlabel left=50:top=50 end create END CREATE SetWindowLong(Form.Handle, -8, 0) SetWindowLong(Application.Handle, -8, Form.Handle) for i = 1 to 10 **woord=woord+str$(i)+chr$(13)** next i lab1.caption=woord Form.ShowModal **sub newaccount** **woord=""** for i = 50 to 60 step+2 **woord=woord+str$(i)+chr$(13)** next i **lab1.caption=woord** **end sub**	Start the loop from 1 to 10. Add the value of i to the **woord** string. CHR$(13) means next line. Sub program for when the user clicks on the window screen. Clear the **woord** string. Start the loop from 50 to 60 with interval counting of 2. Add the value of i to the **woord** string. CHR$(13) means next line.

EXIT FOR. Let's say that you are using a **FOR ... NEXT** function to search for a name in a database. The database is 100'000 lines to search from. The actual name that you are looking for is at entry line **985**. It is therefore a waste of time to search the rest of the database. **The program must exit once the match was found. To do this we use the** EXIT FOR **command on the condition**.

Commands	Description
$TYPECHECK ON $INCLUDE <RapidQ2.inc> dim i as single:dim woord as string declare sub endme declare sub newaccount CREATE Form AS QFORM Caption = "Click Screen":Width = 640: Height = 480:center onclick=newaccount create lab1 as qlabel left=50:top=50 end create END CREATE SetWindowLong(Form.Handle, -8, 0) SetWindowLong(Application.Handle, -8, Form.Handle) for i = 1 to 10: woord=woord+str$(i)+chr$(13):next i:lab1.caption=woord Form.ShowModal **sub newaccount** woord="" for i = 50 to 60 step+2 woord=woord+str$(i)+chr$(13) **if i = 56 then** exit for next i lab1.caption=woord **end sub**	If **i=56** then exit the **FOR** loop. This is why the value of **WOORD** is only till **56**.

9.1.6.2 WHILE ... WEND

The syntax:

While ... ? –
...++
Wend

These commands works in principle the same as the FOR NEXT commands, but a condition is tested before entering the loop. The ... represents the numeric variable used to position the loop process. The ? is the evaluation method used (= >< etc.) The – represents the value tested against. The ... within ...+++ command is again the numeric variable used to position the loop process. The ++ instructs the program to add one to the numeric variable. With this example, enter any of the names within the list to the right and click the button.

Commands	Description
$TYPECHECK ON $INCLUDE <RapidQ2.inc> declare sub displaynames **data elmarie,hendrik, bruce,schutte, sny, struis** dim i as single dim names as string **dim members(10) as string** **for i = 0 to 5:read members(i):next i** CREATE Form AS QFORM Caption = "Click Screen":Width = 640 Height = 480:center create okbutton as qbutton left=100:top=80:caption="Friends onclick=displaynames end create create nameinfo as qedit taborder=0:top=50:left=80:width=100 end create create remark as qlabel top=50:left=400 end create END CREATE SetWindowLong(Form.Handle, -8, 0) SetWindowLong(Application.Handle, -8, Form.Handle) for i = 0 to 5:remark.caption=remark.caption+members(i)+chr$(13):next i Form.ShowModal sub displaynames **i=0** **while i<6** **if lcase$(members(i))=lcase$(nameinfo.text) then showmessage "entry: "+str$(i)** **i++** **wend** end sub	Data list of names String variable for names. FOR LOOP to read and record names within the members list. **Reset i** numerical variable If is less than **6** then Condition if **names are same**. Add **i+1** till larger than **6**. Repeat actions from **while**.

To exit the loop even before the condition has been reached, add the **EXIT WHILE** command to the condition when found.

```
if lcase$(members(i))=lcase$(nameinfo.text) then showmessage "entry: "+str$(i):exit while
```

9.1.6.3 DO ... LOOP

The syntax:

```
Do
...++
Loop until ...?
```

These commands by principle are the same as **WHILE** ... **WEND**. The condition to stop the loop as with **LOOP** and not the starting **DO**.

Commands	Description

$TYPECHECK ON
$INCLUDE <RapidQ2.inc>
declare sub displaynames
data elmarie,hendrik, bruce,schutte, sny, struis
dim i as single:dim names as string:dim members(10) as string
for i = 0 to 5:read members(i):next i
CREATE Form AS QFORM
  Caption = "Click Screen":Width = 640: Height = 480:center
  create okbutton as qbutton
  left=100:top=80:caption="Friends: onclick=displaynames
  end create
  create nameinfo as qedit
  taborder=0:top=50:left=80:width=100
  end create
  create remark as qlabel
  top=50:left=400
  end create
END CREATE
SetWindowLong(Form.Handle, -8, 0)
SetWindowLong(Application.Handle, -8, Form.Handle)
for i = 0 to 5:remark.caption=remark.caption+members(i)+chr$(13):next i
Form.ShowModal
sub displaynames
i=0
do
  if lcase$(members(i))=lcase$(nameinfo.text) then showmessage "entry: "+str$(i)
  i++
loop until i=6
end sub
	Start the loop
	
	End the loop when condition i=6.

To exit the loop even before the condition has been reached, add the **EXIT DO** command to the condition when found.

```
if lcase$(members(i))=lcase$(nameinfo.text) then showmessage "entry: "+str$(i):exit do
```