KBASIC

Tutorial

Version of 2 April 2012

A KBasic course

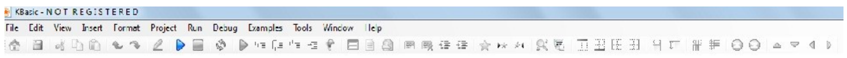
Introduction

New versions of the programming language Basic support structured modular programming which has become a prerequisite for writing correct programs. Notions such as “top-down” and “bottom-up” programming as well as “event driven” program development have become part of the information vocabulary. This tutorial is devoted to teach how to program according to the mentioned principles. It is intended to be used together with a computer equipped with “Kbasic”,[1] not only theoretical but also practical abilities have to be developed.

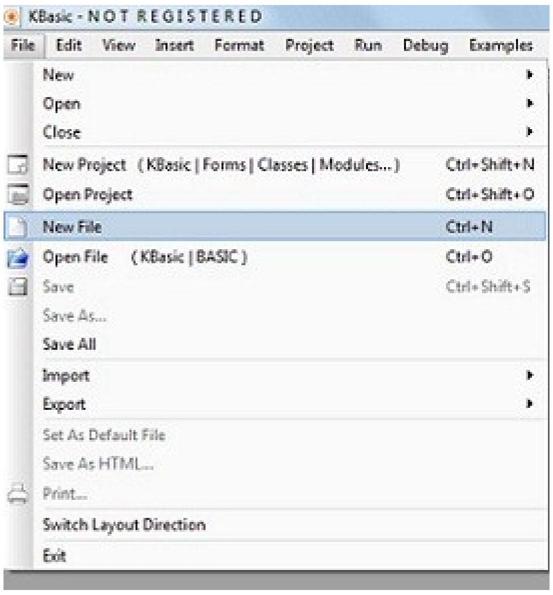
Starting KBasic

Using the trial option, KBasic can perform without the need to pay for it. Later when you decide to continue to use it you should register and pay. It is not too expensive, furthermore you can later decide to upgrade to a professional version, that is a bit expensive, but offers more features. For beginners the professional features are definitely not necessary. First one has to learn how to use the different elements of the Kbasic language and how to use them, later “forms” and what can be placed on them will be an important part of the tutorial. Structural programming will be taken into consideration first, later the event driven concept will be expounded.

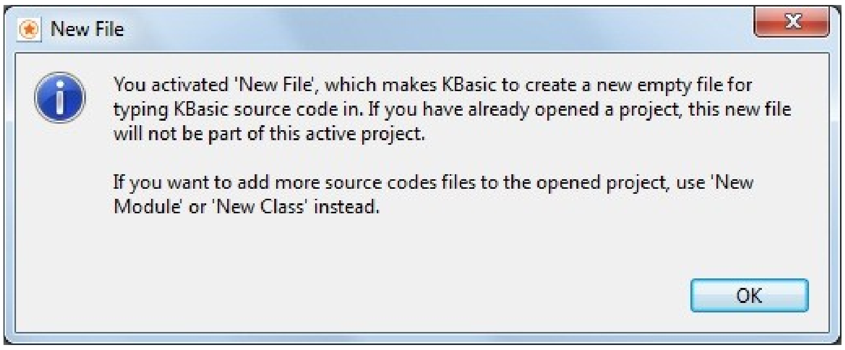
When you start KBasic you will see after clicking “trial” the opening window.The upper part is shown below.


KBasic - NOT REGISTERED
File Edit View Insert Format Project Run Debug Examples Tools Window Help

That is really an overwhelming collection of icons. We choose from “File” the item “New File”. Note that we could have typed Ctrl+N.


KBasic - NOT REGISTERED
File Edit View Insert Format Project Run Debug Examples
New
Open
Close
New Project (KBasic | Forms | Classes | Modules...) Ctrl+Shift+N
Open Project Ctrl+Shift+O
New File Ctrl+N
Open File (KBasic | BASIC) Ctrl+O
Save Ctrl+Shift+S
Save As...
Save All
Import
Export
Set As Default File
Save As HTML...
Print...
Switch Layout Direction
Exit

Surprisingly a warning is displayed. You must get accustomed to the fact that Kbasic likes to inform its users about what it considers important.


New File

You activated 'New File', which makes KBasic to create a new empty file for typing KBasic source code in. If you have already opened a project, this new file will not be part of this active project.

If you want to add more source codes files to the opened project, use 'New Module' or 'New Class' instead.

After clicking “OK” we can insert our code.

The first program will be one that draws a square. As “KBasic” does not know how the square has to be drawn we have to explain in the KBasic language how to do it.2

A computer program is nothing else as a number of prescriptions in computer language: a language the computer understands, in this case BASIC, describing how a certain task should be performed. So in a number of prescriptions it has to be explained how the rectangle should be represented on the screen. The word “prescription” has a certain notion in human language: the practitioner prescribes what to do in case of an illness to get healthy again. A computer program is the same, in a sense as it describes which actions have to be taken successively.

In BASIC different types of “prescriptions” exist. The first we work with is called “subroutine”. A subroutine describes how a certain task has to be performed. Imagine you have to explain to a child how a square has to be drawn. So the first task is: draw a square. We have to explain what is meant by “draw a square”: how to do it. We would say:” draw a horizontal line”, then “draw the two vertical ones” and after that “draw a horizontal line” again. To help the child we could write this down on a piece of paper as follows:

draw square:

draw horizontal line

draw vertical lines

draw vertical lines

draw vertical lines

draw horizontal lines

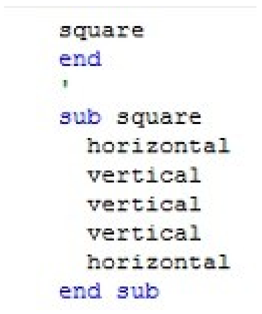
draw horizontal line:

draw "----------------"

draw vertical lines:

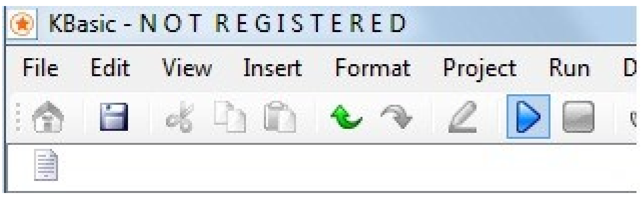
draw “| |”

By doing this we have already nearly written the BASIC program. We enter code that will be our “square program”. So we enter the first lines of that program:



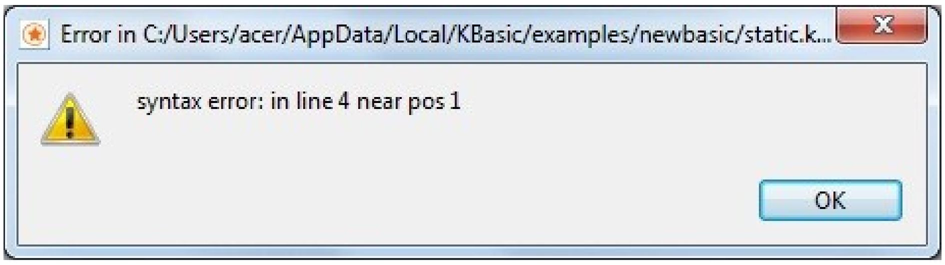
```
square
end
'
sub square
  horizontal
  vertical
  vertical
  vertical
  horizontal
end sub
```

The first line is not a comment line but a command. It says execute the subroutine with the name “square”. The suboutine starts with “sub” followed by its name and ends with “end” followed by it name. Note that you don't need to state in KBasic explicitly that you want a subroutine to be executed. Do not use the word “call”.
We can try to execute the program thus far by clicking the little green triangle.



It is green but turns to blue when clicked.

The result: an error message is displayed:

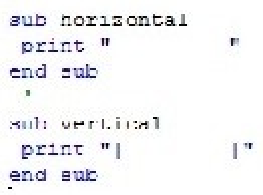


Error in C:/Users/acer/AppData/Local/KBasic/examples/newbasic/static.k...

syntax error: in line 4 near pos 1

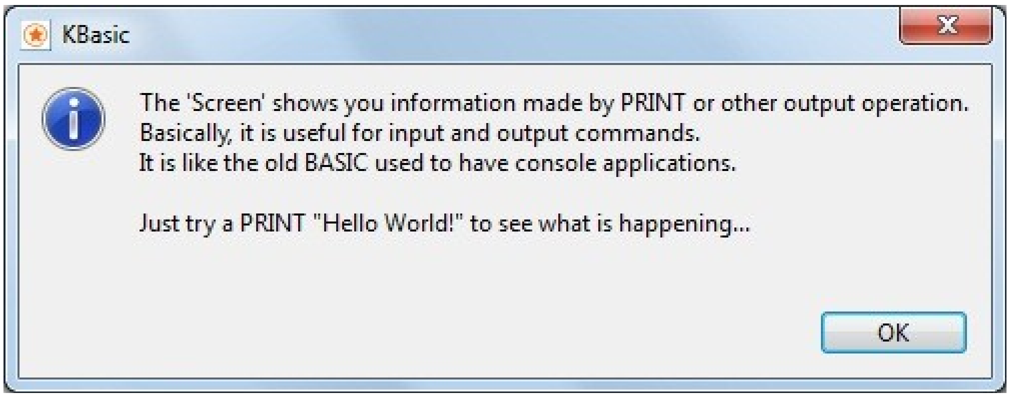
The offending line has changed its colour. It says a syntax error, but the word horizontal is correct, so what is wrong?

“horizontal and “vertical” are subroutine calls. But we did not yet explain to the computer what is meant by these words. So we add the following explanation of what is meant by “horizontal” and “vertical”:



```
sub horizontal
 print "          "
end sub
 '
sub vertical
 print "|         |"
end sub
```

After inserting the above text running the program will result in a window showing a message:

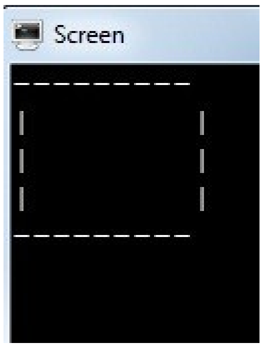


KBasic

The 'Screen' shows you information made by PRINT or other output operation.
Basically, it is useful for input and output commands.
It is like the old BASIC used to have console applications.

Just try a PRINT "Hello World!" to see what is happening...

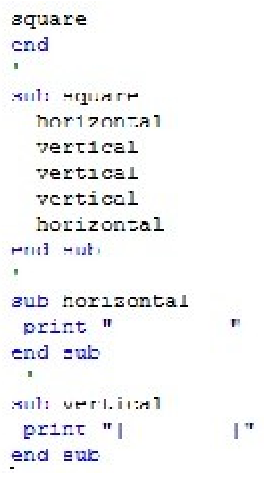
We will not follow this suggestion but click OK after which a new message appears telling us to save the source code first. After obeyed a window will open with the square drawn:



```
---------
|        |
|        |
|        |
---------
```

Unfortunately Kbasic contains a bug so that it is not possible to close the window by clicking the cross in the upper right corner. But it does not impede further working.

The complete square program now follows, it is stored in the listings as “square1”.



```
square
end
'
sub square
  horizontal
  vertical
  vertical
  vertical
  horizontal
end sub
'
sub horizontal
 print "         "
end sub
 '
sub vertical
 print "|        |"
end sub
```

Note that the second line contains the command “end” Do not confuse this “end” with the “end” in “end sub”. After having executed the line containing “End Sub” it returns to the main program. After “square” has been executed we indicate that the program should stop execution by inserting the BASIC word “end”. Words like “sub” and “end” are what are called reserved words They are reserved for BASIC and should not be used otherwise.

We will discuss the routines “horizontal” and “vertical”.

Question

What have the statements contained in these subroutines in common?”

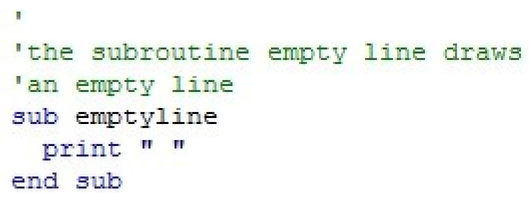
Answer

In both is what has to be written to the screen surrounded by the ” symbol. What is contained between these symbols is called “string”.

Remark

A string may contain also spaces. A space is not nothing.   We changed the source code, we save our work by clicking: “Save all”.

Modify "square" in such a way that the square to be printed
will appear a bit lower on the screen. To do so we add
"emptyline" and explain what is meant by "empty line":
(Stored as emptyline)



```
'
'the subroutine empty line draws
'an empty line
sub emptyline
  print " "
end sub
```

It is advisable to add text to the program that explains to the user its meaning. Short explanations may follow the line immediately. Longer text can be added on separate lines. To indicate that the text is intended to be read by the user and not by the KBASIC system the text should be preceded by the symbol here shown between asterisks: *'* . Also "rem" can be used to achieve the same result, although it is outdated. The term for such lines is "comment". The complete listing can be found as "square" under "Listings" at the end of this tutorial

Note that the different routines of the program "square" are separated by a line containing an empty comment. Furthermore it is advisable to indent the contents of the different subroutines, especially in longer KBASIC programs it is advisable to do so to make them more readable[7].

Exercise

Change the program in such a way that it will show three rectangles

A program that prints your address.

It is possible to perform such a task with a word-processor, but to do so in Kbasic gives you the possibility to exercise the earlier mentioned programming principles. The data used here are those of "Emilie Sagario", you should use your own data, of course.

Again we start with a simple program that we expand into one that will perform as intended. This is called "refinement", that is to say adding levels to the program until arriving at the level of BASIC instructions.

To begin with, what should appear on the screen is as follows

EMILIE SAGARIO

Adamville Compound

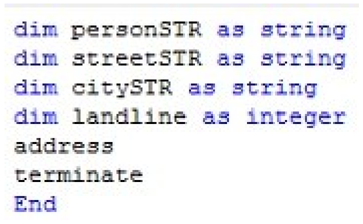
Marigondon, LapuLapuCity

0963848889

This should appear a few lines from the top of the screen

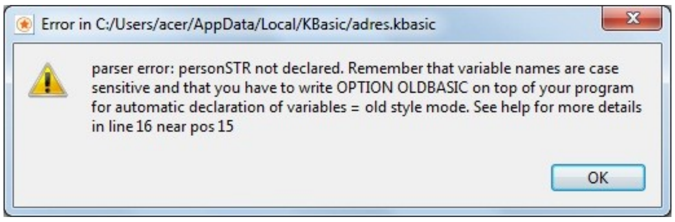
Exercise

Write the “body” of the program.



```
dim personSTR as string
dim streetSTR as string
dim citySTR as string
dim landline as integer
address
terminate
End
```

Note that the information is stored in variables: person STR, streetSTR enz. These variables should be declared: Mentioned in dim(ension) statements. Failure to do so leads to the errormessage:



```
Error in C:/Users/acer/AppData/Local/KBasic/adres.kbasic

parser error: personSTR not declared. Remember that variable names are case
sensitive and that you have to write OPTION OLDBASIC on top of your program
for automatic declaration of variables = old style mode. See help for more details
in line 16 near pos 15
```

If you write the dim-statement as:

Dim personSTR, streetSTR, CitySTR as String

only CitySTR will be stored as a string type. Therefore each dim statement ends with mentioning the type of the variable. This is not obligatory but wise to do as it protects you from making difficult to find errors. It is not necessary to put every variable on a separate line as shown in this example:

dim personSTR as string,streetSTR as string,citySTR as string,landline as integer

Now we can finish the program. It is stored as “adres1” in the listings. Note that the word “landline” has been used. We could have used the term “telephone” as well.

Exercises:

- Modify this program in such a way that it shows your address three times, well separated by a few empty lines.
- Write a program that fills the screen with stars: “*”. Fore example 20 lines of 50 stars
- Write a progam that displays: “Hello“, followed by your name.

A bit of arithmetic and how to compare

Until now we only solved very simple programming problems, such as showing a rectangle and an address. Now we will start writing programs to solve problems involving arithmetical calculations.

The first one adds 2 pesos to the price of a certain commodity.

Here is the “body” of the program:

```
Dim price As Short
Dim priceSTR As String
Const addition As Short = 2
'
newprice
end
'
```

Note that addition is declared as constant:

Const addition as Short = 2

Using a constant instead of a variable like “price” has the advantage that “addition” cannot accidentally be changed.

The subroutine that does the computation is as follows:

```
Sub calculate
 price = price + addition
End Sub
```

The subroutine “Calculate” first gives the variables “price” its value. After that “price” is calculated. First to “price” being 25 the value 2 is added, after that the calculated value is assigned to the variable left of the “=” symbol.

Note that in this case the old value of “price” is no longer available: it has been overwritten.

The program will be expanded in such a way that the user can enter the price. To do so the subroutine “Ask_Price” is added:

```
sub ask_price
  input "Enter a price"; priceSTR
  price=val(priceSTR)
 end sub
 '
```

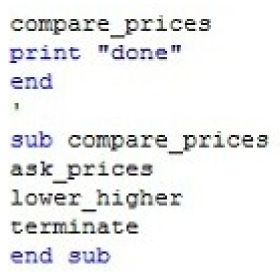
A text is displayed explaining the user what is expected. Everything between the “ symbols is displayed, after that the computer waits for “priceSTR”. It is converted to “price” by using the function “val” as shown.

More about functions later.

The subroutine “Ask_Price” may also be named “ASK_PRICE”, so names may be given in lower as well upper case. It is allowed to include the symbol “_” in names. But the KBasic literature does not support this option.

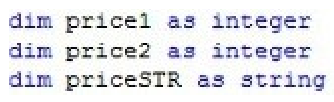
The complete program is stored as price1.

A program has to be written that compares two prices, input by the keyboard and then states which one is lowest and which is highest. In it most simple form the start of the program is as follows:



```
compare_prices
print "done"
end
'
sub compare_prices
ask_prices
lower_higher
terminate
end sub
```

Two integervariables are used in this routine: “price1 and “price2”. Furthermore one string-variable: priceSTR. They are declared at the beginning of the program as shown below.



```
dim price1 as integer
dim price2 as integer
dim priceSTR as string
```

Explain what has to be done in the subroutine “ComparePrices”.

The following question has to be answered: “Is the first price that has been typed lower than the last one?” If yes the output should be: “The first price is lower than the second”, else: “The first price is higher than the second”. Important words used here are “if” and “else” as these express what should be done under which condition. In this case what has to be printed on the screen.

If prijs1 < prijs2 then lower else higher end if

This is called a conditional statement. After if is stated what has to be done when the condition is true, after else in case it is false. Using more words the statement could be read as follows: If it is true that prijs1 is lower than prijs2 than the subroutine lower should be executed else higher. See how this is coded in Kbasic:

```
Private Sub lower_higher
If price1 < price2 Then
 lower
Else
 higher
End If
End Sub
```

Now write the complete program. Write first “lower” and “higher”

```
Private Sub lower
  print ”price 1 is lower than price 2”
End Sub
  '
Private Sub higher
  print ”price 1 is higher than price 2”
End Sub
```

Write the subroutine “terminate” .

```
Sub terminate()
  print ””
  input ”Enter a character to stop; priceSTR
  price1 = val(priceSTR)
  End
```

```
End Sub
```

The finished program is stored in the listings as “price2”.

Test what happens when two identical prices are entered.

When the same price is entered the program erroneously prints that price1 is higher than price 2 because price1 is not lower than price2 so what follows after “else” is done.

Modify the program in such a way that this possibility is handled correctly. The subroutine “LowerHigher” has to be adapted and “equal” has to be added:

```
Private Sub lower_higher
  If price1 = price2 Then
   equal
  Else
   If price1 < price2 Then
    lower
   Else
    higher
   End If
 End If
End Sub
  '
Sub equal
print ”price 1 is as high as price 2”
End Sub
```

Exercises

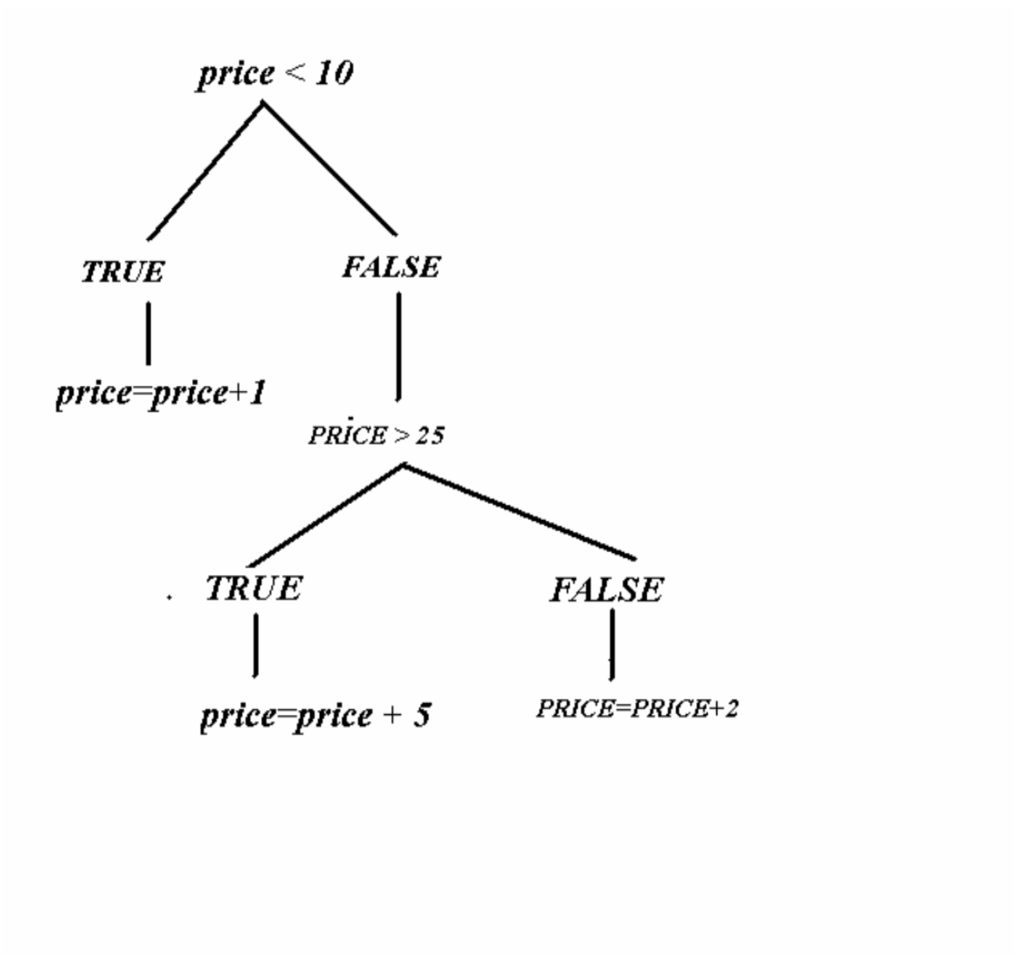
Write a program that asks the user to enter a price. After that the program determines whether that price is higher or lower than 25. Prices below that value should stay the same. To prices of 25 and higher an amount of 2 should be added. The new price is shown then.

Write a program that asks the user to enter a price. After that the program determines whether that price is higher or lower than 10. To prices below that value an amount of 1 should be added. For prices between 10 and 25 the amount is 2. Prices higher than 25 should increase by 5.

As the last exercise is more difficult than the former, it will be discussed in detail. Before writing the necessary “if”-statements it is wise to first write down a table and after that a decision tree to show what has to be done.

Price lower than 10	price between 10 and 25	price higher than 25
add 1	add 2	add 5

The decision tree looks as follows:



Now we can write the “IF”-statements:

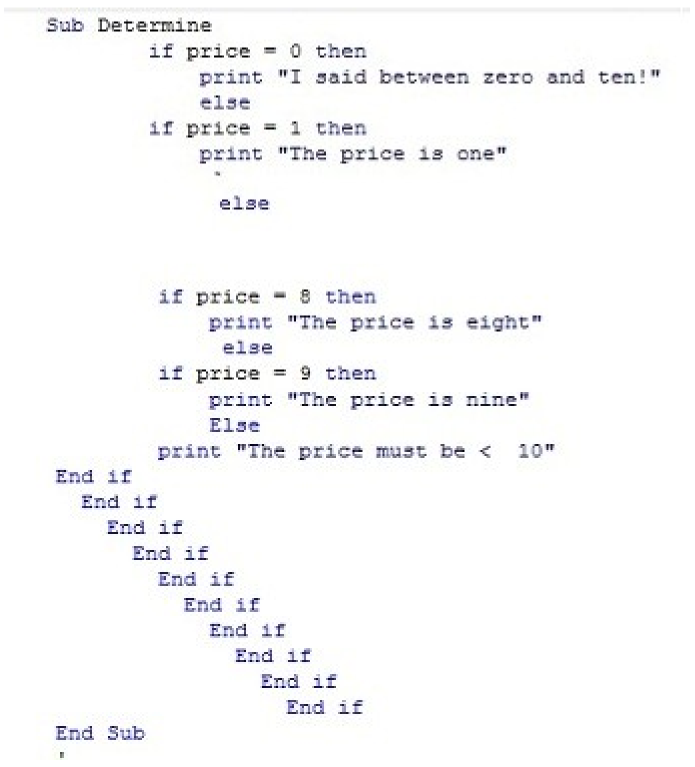
```
Private Sub calculate
If price < 10 Then
 higher1
Else
If price > 25 Then
 higher5
Else
 higher2
End If
End If
End Sub
```

Note that in the subroutine “calculate» the text END IF” has been used twice. This is necessary as each “IF”-statement should have its accompanying “END IF”. Failing to do so will generate an error message. Take note of the indentations to keep the code easy readable. You can now type the program displayed above in KBasic to experiment with it. (The complete program is stored as “price3”)

Exercise

Write a program that accepts a price between zero and ten.

The most important subroutine is the one that determines what to print: The program should display which number has been entered.



```
Sub Determine
        if price = 0 then
            print "I said between zero and ten!"
            else
        if price = 1 then
            print "The price is one"
            '
             else


         if price = 8 then
             print "The price is eight"
              else
         if price = 9 then
             print "The price is nine"
             Else
         print "The price must be <  10"
 End if
   End if
     End if
       End if
         End if
           End if
             End if
               End if
                 End if
                   End if
 End Sub
 '
```

This was rather cumbersome to have to write all those “IF”s with the accompanying END IF's. Easily you forget one of the “end if”s and an error message will be generated. Later we will learn that it can be done easier. The complete program is stored as “manyif”s.

Working with characters

In computer science “character” is not about a characteristic property of a person, but about the input that can be given using a keyboard. These can be divided in letters (a – z) ciphers (0 – 9) and special characters such as “!”, “@” etc. All together are called alphanumeric characters. Everything you see on the keyboard, so to speak.

To distinguish between values assigned to characters to be manipulated after being input by the keyboard and values used for computing the name of the variable used for such manipulation has to be dimensioned as string as has already been shown several times in the preciding programs.

It has become customary by BASIC programmers to have the name of such a variabele be preceded by “str” (It is called hungarian notation as it was proposed by a hungarian programmer). Some programmers use this string as a postfix instead of a prefix. It is up to you to follow one of these customs to improve the readability of your code. In this tutorial the suffix “str” has been capitalised for that purpose.

If the program contains the DIM-statement “price as string”, the price is stored in a different way than when the line would read “ dim price as short”.

To help the programmer in manipulating characters he can use what are called character functions. Later the notion “function” will be explained in extenso, for the moment it suffices to now that a function returns a value, that can be used in the program.

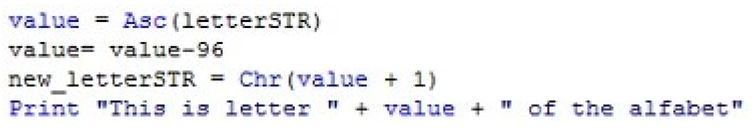
Some of the functions available in Kbasic to manipulate characters are slightly different from those as used in most BASIC dialects. We will discuss now these functions as defined for Kbasic. These functions are:

CHR(n) - return character of ascii value n STR(n) - returns string equivalent of n.

Example:

When a letter is typed, for example “c” the sentence “this is letter 3 of the alphabet” will be shown on the screen.

As an exercise write this program first and after that look at the solution.
Part of the subroutine “determine” is shown below:



```
value = Asc(letterSTR)
value= value-96
new_letterSTR = Chr(value + 1)
Print "This is letter " + value + " of the alfabet"
```

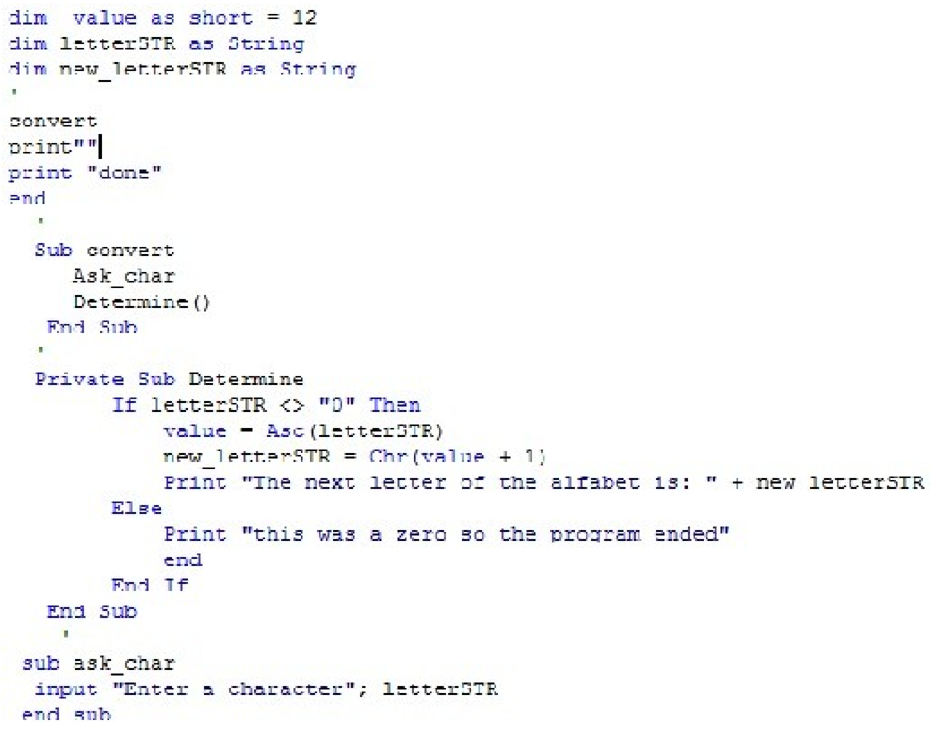
Because the value assigned to the letter a is 97, we need to deduct 96 from this value to get the correct number to be displayed.

The program is stored as “whichar”

Exercise:

Modify the program in such a way that the next letter from the alfabet is displayed.

The solution is shown below: (stored as nexchar)



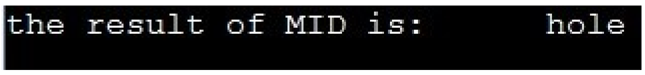
```
dim  value as short = 12
dim letterSTR as String
dim new_letterSTR as String
'
convert
print""
print "done"
end
  '
  Sub convert
     Ask_char
     Determine()
   End Sub
  '
  Private Sub Determine
        If letterSTR <> "0" Then
            value = Asc(letterSTR)
            new_letterSTR = Chr(value + 1)
            Print "The next letter of the alfabet is: " + new letterSTR
        Else
            Print "this was a zero so the program ended"
            end
        End If
   End Sub
    '
 sub ask_char
  input "Enter a character"; letterSTR
 end sub
```

Stored as “nexchar”

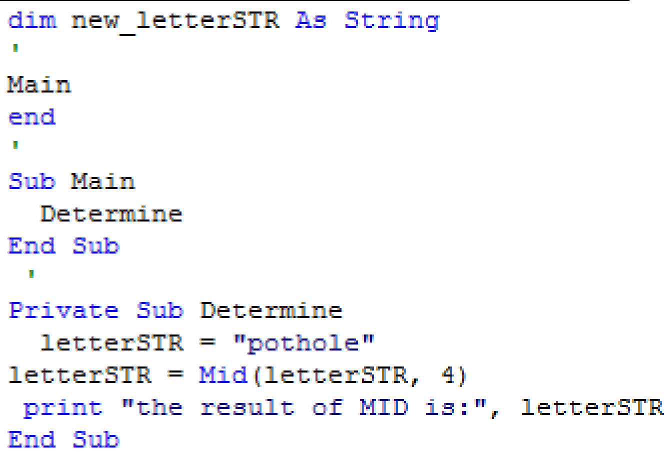
ASC(sSTR) - ascii value of sSTR

Example as above .

MID -The function MID as implemented in Kbasic is shown in the following example:



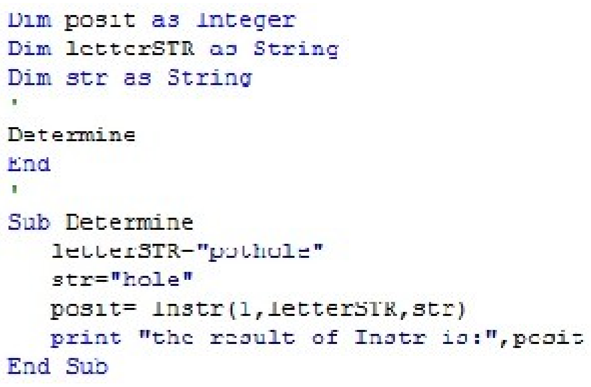
```
the result of MID is:      hole
```



```
dim new_letterSTR As String
'
Main
end
'
Sub Main
  Determine
End Sub
 '
Private Sub Determine
  letterSTR = "pothole"
letterSTR = Mid(letterSTR, 4)
 print "the result of MID is:", letterSTR
End Sub
```

So the function returns the string starting at the fourth character up to the end.

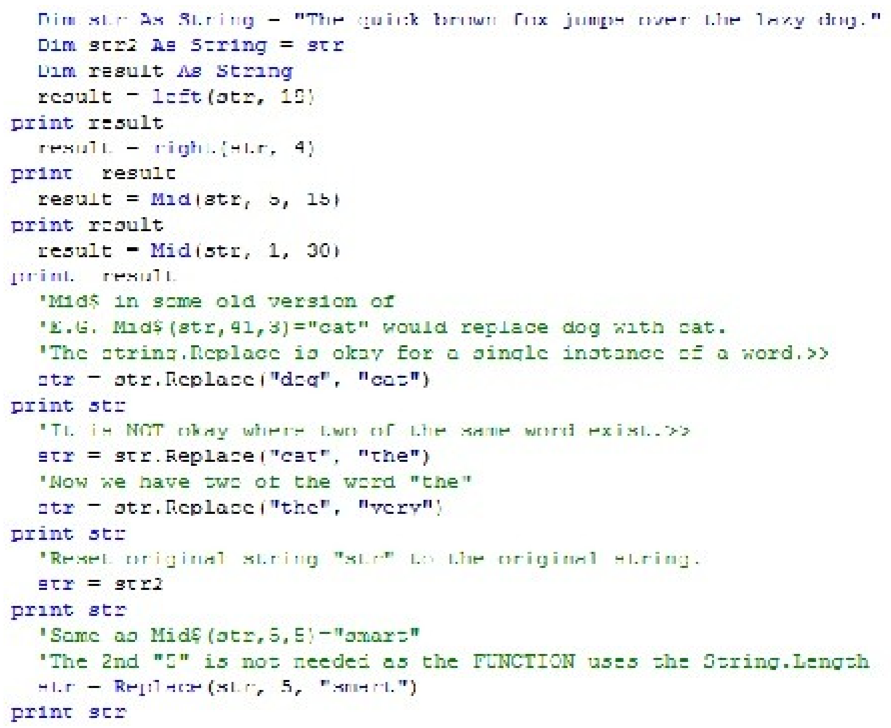
INSTR - The following example show how “Instr” should be used:



```
Dim posit as Integer
Dim letterSTR as String
Dim str as String
'
Determine
End
'
Sub Determine
   letterSTR="pothole"
   str="hole"
   posit= instr(1,letterSTR,str)
   print "the result of Instr is:",posit
End Sub
```

The variable “posit” is an integer type. It will contain the position of “str” in “letterstr”. So this code will deliver the value four as it is the position of the second argument within the first.

The functions left, right, mid and select are demonstrated in the following program



```
  Dim str As String = "The quick brown fox jumps over the lazy dog."
  Dim str2 As String = str
  Dim result As String
  result = left(str, 15)
print result
  result = right(str, 4)
print  result
  result = Mid(str, 5, 15)
print result
  result = Mid(str, 1, 30)
print  result
  'Mid$ in some old version of
  'E.G. Mid$(str,41,3)="cat" would replace dog with cat.
  'The string.Replace is okay for a single instance of a word.>>
  str = str.Replace("dog", "cat")
print str
  'It is NOT okay where two of the same word exist.>>
  str = str.Replace("cat", "the")
  'Now we have two of the word "the"
  str = str.Replace("the", "very")
print str
  'Reset original string "str" to the original string.
  str = str2
print str
  'Same as Mid$(str,5,5)="smart"
  'The 2nd "5" is not needed as the FUNCTION uses the String.Length
  str = Replace(str, 5, "smart")
print str
```

The program is stored as “charfuncs”.

Suppose “TheString » = « The quick brown fox jumps over the lazydog. »

The space between “lazy” and “dog” is left out purposely. Insert it in the demonstration program to see the result.

As an exercise modify the program that demonstrates the function “asc” in such a way that it asks for the input of one letter and will print on the screen the next character and its ascii value.

Until a value has been found

We will change the character programs discussed, in such a way that the user is not confined to input only one letter. When he enters a zero the program stops. Change the program that prints the next letter of the alfabet in this way. Kbasic offers diiferent commands to chosen from. They are respectively:

WHILE – END WHILE

DO LOOP – WHILE

DO WHILE – LOOP

DO LOOP – UNTIL

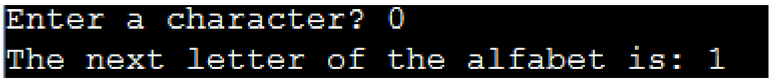
DO UNTIL – LOOP

If we allow the user to input a letter only a certain number of times the command FOR-NEXT, which will be discussed later, can be used, in that case the user does not need to input a value of zero to stop. First we use the WHILE-END WHILE command. Look at the program "price2". This time it is not the module "Determine" that has to be adapted, but "convert". Convert has to be done until a zero has been typed. This module now becomes:

```
Private Sub convert
 While letterSTR <> "0"
  Ask_Letter
  Determine
 End While
End Sub
```

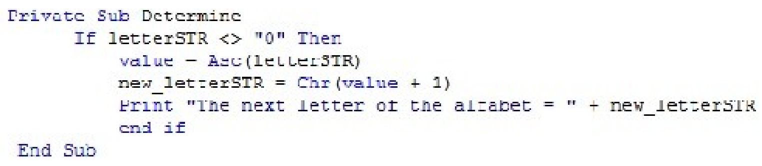
After "while" is the condition stated that stops execution of the commands. The complete program can be found in the listings under "while".

Now if we execute this program we will see that the program prints after entering the value of zero:



```
Enter a character? 0
The next letter of the alfabet is: 1
```

This should not happen of course. Look at the subroutine "determine" and determine what is wrong.



```
Private Sub Determine
      If letterSTR <> "0" Then
          value = Asc(letterSTR)
          new_letterSTR = Chr(value + 1)
          Print "The next letter of the alfabet = " + new_letterSTR
          end if
 End Sub
```

Something is missing. How can it be remedied?

Answer

We could add an IF-statement in the subroutine "determine":

Solution:

```
 Private Sub Determine
        If letterSTR <> "0" Then
            value = Asc(letterSTR)
            value = value - 96
            new_letterSTR = Chr(value + 1)
            Print "This is letter: " + value + "of the alfabet"
        Else
            Print "this was a zero so the program ended."
        End If
    End Sub
```

Perform the same task in such a way that the DO UNTIL – LOOP construction is used.

```
Private Sub convert
  Do Until letterSTR = "0"
   ask_char
   Determine
  Loop
End Sub
```

```
'
```

Note that only the condition stated differs. It depends on the problem to be solved which construction should be used.

Instead of testing wheter “letterSTR” equals zero we could the same using the variable “value”. As an exercise chang the subroutine “convert” accordingly.

Solution:

```
Private Sub convert
 Do Until value = 0
   ask_char
  Determine
 Loop
End Sub
'
```

Run the program and note that it does not function as intended as immediately “done” is displayed. Why?

Solution:

The program tests whether “value” equals zero and because Kbasic initilises variables to zero the program behaves accordingly. This can easily be remidied by giving “value” an initial value:

dim value as short = 5

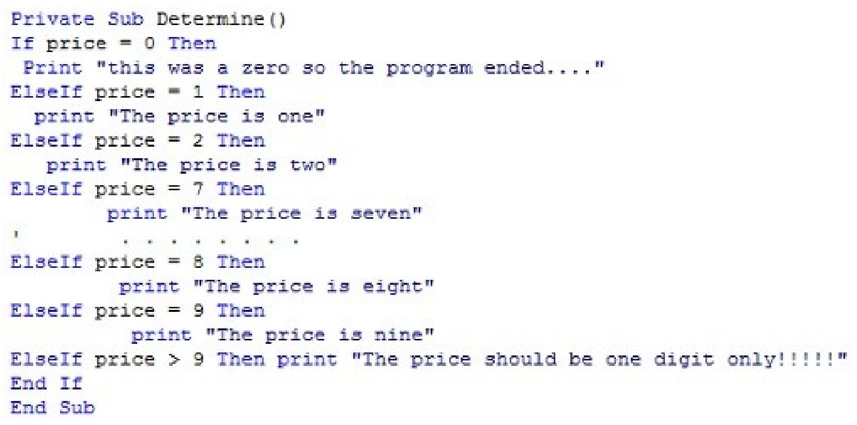
Any value > 0 will do. This solves the problem The program is stored as “chnot” .

It is left to the user of this tutorial to apply the other loop-constructs. For example to practise with what has been discussed above write a program that asks for the input of one letter and prints on the screen the numerical value assigned to it.

As has been said earlier there is another way to perform a certain task. It is called the FOR-NEXT loop. This construction can be used when a task has to be performed a certain number of times. For example, we want a program that converts a digit into a string. Without the possibility of using the FOR-NEXT loop, we have to write a lot of statements such as in the program that converts a digit to a string:

```
Private Sub Determine()
If price = 0 Then
 Print "this was a zero so the program ended...."
ElseIf price = 1 Then
  print "The price is one"
ElseIf price = 2 Then
   print "The price is two"
ElseIf price = 7 Then
        print "The price is seven"
'        . . . . . . . .
ElseIf price = 8 Then
         print "The price is eight"
ElseIf price = 9 Then
          print "The price is nine"
ElseIf price > 9 Then print "The price should be one digit only!!!!!"
End If
End Sub
```

This is really not a pleasant way of programming, furthermore the many ifs sould be accompqnied by the same amount of end ifs. We can improve upon it a little bit as shown below:


```
Private Sub Determine()
If price = 0 Then
 Print "this was a zero so the program ended...."
ElseIf price = 1 Then
  print "The price is one"
ElseIf price = 2 Then
   print "The price is two"
ElseIf price = 7 Then
        print "The price is seven"
'        . . . . . . . . .
ElseIf price = 8 Then
         print "The price is eight"
ElseIf price = 9 Then
          print "The price is nine"
ElseIf price > 9 Then print "The price should be one digit only!!!!!"
End If
End Sub
```

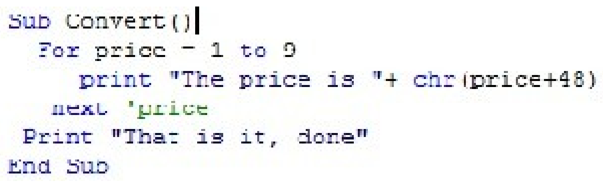
Still a lot of elseifs. Later we will rewrite the program using the CASE construction in such a way that all the digits between 0 and 10 are converted to a string and printed.

Exercise

Rewrite the above program in such a way that the text ”The price is” followed by the digit is printed.

Solution

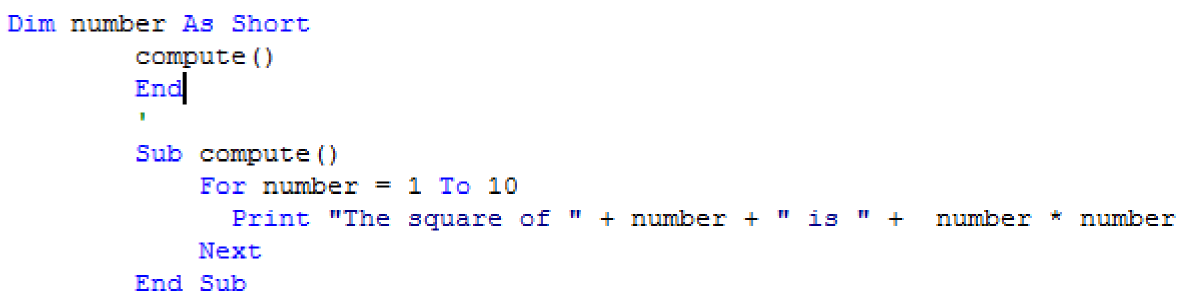
Only the subroutine “convert” has to be rewritten:


```
Sub Convert()
  For price = 1 to 9
     print "The price is "+ chr(price+48)
   next 'price
 Print "That is it, done"
End Sub
```

Exercise

Use the FOR-NEXT construction to rewrite the program “square”

Solution


```
Dim number As Short
          compute()
          End
          '
          Sub compute()
               For number = 1 To 10
                 Print "The square of " + number + " is " +  number * number
               Next
          End Sub
```

Exercise

Write the “price” program using a loop construct.

Solution

The subroutine convert has been written as follows:

```
Sub convert
  Do Until value = 0
    Ask_char
    Determine
  Loop
End Sub
```

Try it in your program and note that the program doesn't function at all.

Question

What is the cause of the problem.?

Answer

When starting a program all numerical variables are set to zero.
When the “Do – Until” condition is tested the value of price is zero
and therefore the program ends.

Solution

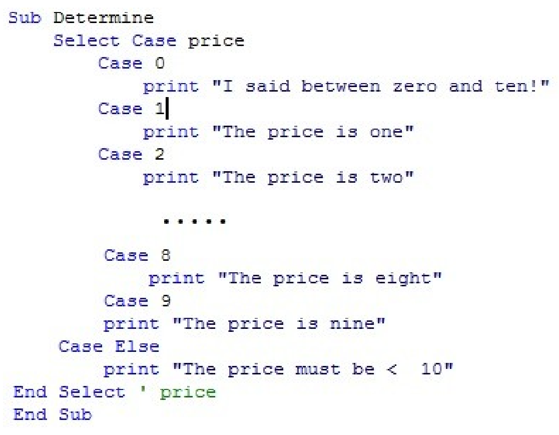
you can overrule the setting to zero by giving variables an initial value as shown below.

```
  Dim price As Short = 12
```

Changing the Dim statement this way solves the problem.
The program is stored as “testchar”.

In case of

A useful construction to simplify a task is offered by the SELECT CASE construction. This can be used, for example, if the user is allowed to enter a value. As it is not known beforehand what the input will be a lot of IF statements would be required that handle all possibilities. Look again at the program that accepts a price between 0 and 10. Here the routine “Datermine” is programmed using the SELECT CASE construction:



```
Sub Determine
    Select Case price
        Case 0
            print "I said between zero and ten!"
        Case 1
            print "The price is one"
        Case 2
            print "The price is two"

              .....

         Case 8
             print "The price is eight"
         Case 9
         print "The price is nine"
     Case Else
         print "The price must be <  10"
 End Select ' price
 End Sub
```

Note that the variable price is obligatory after “Select Case” but added as comment after “End Select” to make clear for the reader end select of what.

The complete program is stored named “selcas”.

Array’s

Suppose we need to decide, after 10 numbers has been input, which of these is the biggest. It would be clumsy when we had to declare ten variables, such as number1, number 2 etcetera. Again there exists a simple solution. We will make an array of these ten numbers. First we have to tell BASIC how big our array will be by means of the dimension statement:

DIM number[10]

Around “10” we could have used the round brackets, but it is advisable to use the quare ones. The program is stored as “tennum”:7

Exercises

A certain Italian Leonardo Pisano, better known as Leonardo Fibonacci (Fibonacci is a reduction from “Filius Bonacci”, meaning son of Bonaccus) put a lot of effort in replacing the Latin numerals (I, V, X, L, C, M) by the Arabian ones (1, 5, 10, 50, 100, 1000) respectively. He poses in his book “Liber Abaci» the following problem: How many pairs of rabbits does one have at the end of a year as at the beginning of that year one has only one pair of rabbits (a pair of rabbits consists of a male an female rabbit) and the following rules apply:

1. Each pair gets two rabbits every month.
2. Rabbits are grown up a month after birth.
3. Rabbits don’t die.

Write a program that shows how the number of rabbits grows. First write down how the number of rabbits grows to get insight how the problem should be solved.

Solution

After 1 month 1 pair is grown up.

After 1 month there is 1 pair of rabbits.

After 2 months 1 pair is grown up and 1 pair of rabbits born in that month.

After 2 months there are 2 pairs of rabbits.

After 3 months 2 pairs are grown up and 1 pair of rabbits born in that month.

After 2 months there are 2 + 1 = 3 pairs of rabbits.

After 4 months 3 pairs are grown up and 2 pairs of rabbits born in that month.

After 4 months there are 3 + 2 = 5 pairs of rabbits.

Etcetera

The numbers indicating how much pairs of rabbits one owns are called the Fibonacci-numbers. These are, as can be read from the above: 1, 2, 3, 5 etcetera.

Inspect the solution stored as "fibon" and note how the wrong output about the first month has been suppressed.

Note the first dim statement:

Dim month as integer = 0

Failure to initialise "month" will generate an error message!

Another problem to be solved is determination of prime numbers. First a textual explanation will be given after that try to solve the problem. Use an array!

First the definition of a prime number will be given: A prime number is a natural number divisible by 1 and itself only. The number 1 is not considered a prime number. The solution described here is an old one. It has been formulated by Eratosthenes. He described the solution as sieving out the numbers that are divisible, for that reason the solution resented here is called the sieve of Eratosthenes.

Copy the following table on paper as it will be used hereafter.

1	2	3	4	5	6	7	8	9	10

11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

We will now sieve out numbers that are not primes by drawing a line through them. As has been said 1 is not considered a prime so we put a line through it. The number 2 is a prime, we indicate it by encircling it.

Which numbers do you think will be removed first?

Answer

All that are divisible by 2. To start with, these are the numbers in the column under 2. To remove them we draw a vertical line starting at 12. Then the numbers that has to be withdrawn are in the column starting with 4. We draw now a vertical line starting at 4, after that we jump over the next column and again draw a line, up until all numbers in the column starting with 10 has been eliminated too.

Note that all numbers divisible by 3 are removed by drawing a ling diagonally. In general the procedure can be stated as: Encircle a prime number and then remove all number that are divisible by them.

Exercise

Write now the program “Sieve of Eratosthenes”, but read first what follows.

In the program to write we can use the BASIC-function MOD. To sieve out a number that is devisable by 2 we write: rest = thisnumber MOD 2. As can be guessed easily the rest of the division of ‘thisnumber’ by 2 is moved to the variable ‘rest’. Now with this information provided write the program. After that look at “Erathos”.

Introduction to working with data

Manipulation of a huge number of data is an old problem. The first magistrate who organised a census was the Roman king Servius Tullius; also according to the bible the Romans held census. How these data were manipulated we don’t know. At the end of the nineteenth century the manipulation of data resulting from census, held in the USA, took so much time that the result became available when a new bookselling was organized. This census took place every ten years. In 1890 the result of the census was 62 million data. A new method to manipulate this enormous amount had to be sought. An American statistic, Dr Herman Hollerith invented an apparatus that enabled to process the data in one third of the time it took to do that for the census of 1880 which were the data of 50 million people. The data of the census of 1890 were punched by hand in thin cards that got the name ‘punch card’, of course. These cards were then fed into a machine that was equipped with electronic contacts. When a hole was found an electric current incremented a counter by one. For every row in the card the same gadget was implemented. Dr Hollerith soon got the idea to apply his invention to administrative processes. The first machines were installed at the American Railways. The next machine that Hollerith developed was a sorting machine. In 1896 Hollerith started the “Tabulating Machine Corporation”. In 1911 two other companies joined and the name was changed to “Computing Tabulating Recording Company”. In 1914 Hollerith’s administrative system consisted as follows:

1. a hand punch machine data made it possible to enter 9 punches at the same time.
2. an electric sorting machine
3. An electric tabulating machine

In 1924 the name was changed again, this time into “International Business Machines Corporation”, well known as “IBM”. This company produced until 1960 machines that processed punch cards. The last one did hundred cards in one minute. However, programming with a computer this machine was not yet possible. It had to be done by inserting cables in a board, in the same way as long ago, telephone calls were made possible.

The first computers are developed at scientific institutions and were used only to do mathematical computations. Again a volkstelling was the impuoes for the next development. J. Prosper Eckert and dr. John Mauchly developed for the Bureau of censuses the “Universal Automatic Computer” shortened to “Univac”. They started the “Eckert-Mauchly Computer Corporation, Later bought by “Remington-Rand” and after that to “Sperry Rand”. The “Univac” is now considered to be the first commercial available computer. One is installed in the “Smithsonian Institue”, at Washington DC. In total 48 of these machines were built.

Working with files

One of the first programs we developed was a program that prints an address. Now suppose that we want to preserve that address, possibly with the addition of more addresses. To make that possible we need to store them in a “file”. A file is a collection of data. These data can be alphanumeric ones such as adresses, but also programs. We are concerned now about storing our alphanumeric data. To store our data we have to create a file. This is done with the following statement. (what is typed between brackets is an example of course)

File.Create("H:\mytest.txt")

Note that the disk used in this example is “H”. It is strongly advised when experimenting with files not to use your “C” drive, preferably a disc such as a “Secure Digital” one.

After having created the file we open it with the following statement:

OPEN "H:\mytest.txt" FOR OUTPUT AS #1

Between the quotes the name of the file is written, in our case mytest.txt. Then we will store the data, so we will “output” it, for each item we have to write an output statement, the first is:

PRINT #1, "A sentence"

The file has to be preserved, this is done by the “close” statement:

Close #1 ' close file

A simple program that stores in a file the data used for the earlier developed program adres is:

```
Dim tekst as string
File.Create("D:\adres.txt")
OPEN "D:\adres.txt" FOR OUTPUT AS #1
print #1,"name:Emilie Sagario|"
print #1,"adres:Adamville Compound|"
print #1,"city:6015 Marigondon, Lapulapu City|"
print #1, "country:Philippines|"
print #1,"telephone:landline: 00 637239474563|"
print #1,""
Close #1
Print "done"
End
```

Because Kbasic contains several bugs when it comes to file handling some measures has been taken to circumvewnt them. The last “Print #1” statement will generate a blank. This is done as the last character written gets lost. Furthermore each statement ends with the character “|", because

Kbasic fails to end the lines with carriage-return characters. In ths way we are able to separate the items printed.

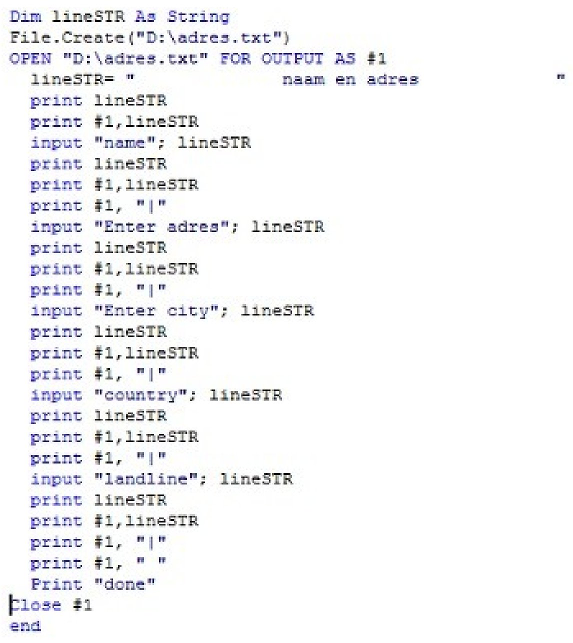
We want to see the contents of the file just created. To do so do so we open the file. The name of the file has to be the same as the one on the disk of course, instead of “emi” we can use any name.

```
' now to read the address:
Dim tekst as string
' open file for reading
Open "D:\adres.txt" For Input As #1
Do While Not EOF(1) ' test for end of file
Line Input #1, tekst ' get the data from file
Print tekst
Loop
Close #1 ' close file
Print "this ends the output program"
```

The information read by the command “`Line Input #1`” will be stored in the variable “tekst” and then shown on the screen. This simple program just displays the line read as it is. It could be stored in an array for further manipulation of course.

As an exercise modify the program in such a way that in the line read the caracter “|” is searched and when found an appriate action is taken.

It would rather cumbersome if we had to change our program every time we have to enter a new address. Better to make it possible to enter those data with the keyboard. The following simple program does it:



```
Dim lineSTR As String
File.Create("D:\adres.txt")
OPEN "D:\adres.txt" FOR OUTPUT AS #1
  lineSTR= "                naam en adres                "
  print lineSTR
  print #1,lineSTR
  input "name"; lineSTR
  print lineSTR
  print #1,lineSTR
  print #1, "|"
  input "Enter adres"; lineSTR
  print lineSTR
  print #1,lineSTR
  print #1, "|"
  input "Enter city"; lineSTR
  print lineSTR
  print #1,lineSTR
  print #1, "|"
  input "country"; lineSTR
  print lineSTR
  print #1,lineSTR
  print #1, "|"
  input "landline"; lineSTR
  print lineSTR
  print #1,lineSTR
  print #1, "|"
  print #1, " "
  Print "done"
Close #1
end
```

(in the listings stored as adres1)

To have a look at our file we can use the earlier developed program but don’t forget to adapt the line to open the file for input.

As an exercise simplify the “adres” program.

A lot more could be said about working with files, but for an introductory course we will not go further into it. 654

Different subroutines

A well written program is made up as a number of units. Until now only a few types have been used: the subroutine and the function.There is another type of subroutine: the one with arguments. First the difference between the simple subroutine, used thus far and the one with arguments will be explained.

Suppose somebody is working for you and does as a rule two tasks: cleaning the house, and going to the supermaket to buy for you what is needed. As it is assumed that cleaning the house always involves the same task, we could say in computer terms: clear_the_house. But shopping is not always about the same items, we have to provide our helper with a shopping list. In computer terms we have to provide a list of items, called the arguments. In our example the shopping task could be invoked by: do_shopping salt, pepper, rice. We willl now write a program using such a list of items.

As an example we change the program that accepts a price between 0 and 10 and then converts the number to text. In the first place we have to change the subroutine calculate:

'vanaf hier

```
Dim price As Integer
Dim newprice as Integer
Const addition as Integer = 2
'
addprice
end
'
Sub addprice
price = 25
calculate(price)
show_price
End Sub
'
Sub calculate(ByVal pretium)
newprice = pretium + addition
End Sub
```

```
'
Sub show_price()
print "the new price = " + newprice
End Sub


'tot hier


Sub calculate(ByVal pretium)
newprice = pretium + addition
End Sub
'
```

After the name of the subroutine the argument, in this case “pretium” follows enclosed in brackets. The routine show_price has been adapted too.

```
Sub show_price()
print "the new price = " + newprice
End Sub
```

It is customary to add a pair of bracketsbehind the name of the subroutine even when the list of arguments is empty.
The calls to the subroutine “calculate has been changed accordingly:

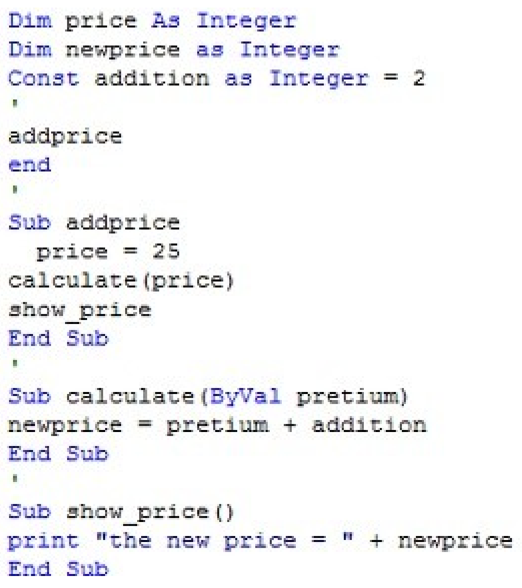
```
calculate(price)
```

We could have written:
calculate(25)

But to make sure that when calling calculate” ”price” should be used and not “pretium”,

Even when not using arguments it is customary to use brackets to indicate that the list of arguments is empty, as for example: show_price()

Now that we know the meaning of brackets behind the names of subroutines we will adhere to that practice. Here follows the complete program for your perusal:



```
Dim price As Integer
Dim newprice as Integer
Const addition as Integer = 2
'
addprice
end
'
Sub addprice
  price = 25
calculate(price)
show_price
End Sub
'
Sub calculate(ByVal pretium)
newprice = pretium + addition
End Sub
'
Sub show_price()
print "the new price = " + newprice
End Sub
```

Look at the subroutine calculate.
The name of the argument has been changed to “pretium” and is preceded by “ByVal”. Use always a different variabele name following the term “ByVal” and use that in the subroutine. It will save you a lot of trouble! “ByVal” means that Kbasic will change the value of price. So after the subroutine has been executed the variable, in this case “price” has changed. If instead of “ByVal”, “ByRef” had been used Kbasic will make a copy of “price” and act upon that copy. So the variable “price” will not have been changed and has the same value after exection of the subroutine as before. There is seldom a need to use “ByRef”, therefore the default text suggested by Kbasic is “ByVal” and will be used as such.

If everything is clear thus far we will now discuss the notion “function” and how to use it. What is the advantage of using a function over the call of the subroutine with parameters? We can use the result of the function call immediately in our program as the result is assigned to the variable used in the call:

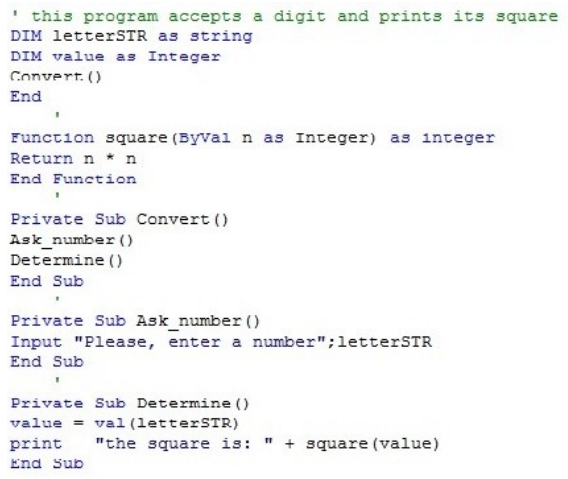
```
print "the square is: " + square(value)
```

Suppose the variable “value” was given the value 5 then the print command prints “25” as the function “square” returned that value

Look now at the function square:

```
Function square(ByVal n as Integer) as integer
  Return n * n
End Function
```

and look at the complete program:



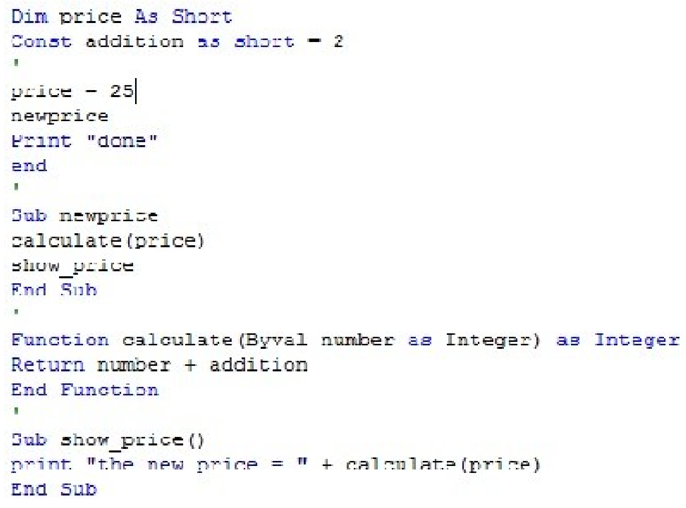
```
' this program accepts a digit and prints its square
DIM letterSTR as string
DIM value as Integer
Convert()
End
    '
Function square(ByVal n as Integer) as integer
Return n * n
End Function
    '
Private Sub Convert()
Ask_number()
Determine()
End Sub
    '
Private Sub Ask_number()
Input "Please, enter a number";letterSTR
End Sub
    '
Private Sub Determine()
value = val(letterSTR)
print   "the square is: " + square(value)
End Sub
```

It is clear what the function does. Note that the variable used in the function, in this case “n”, should not be declared outside the function. The argument(s) have to be provided between “(” and “)” after the name of the function “square”, exactly in the same way as for the subroutine with arguments The program is stored as “funcsquare”.

Exercise

Rewrite the program that adds 2 pesos to the price of 25 using the function calculate.

Try to rewrite the program before looking at the solution given below.



```
Dim price As Short
Const addition as short = 2
'
price = 25
newprice
Print "done"
end
'
Sub newprice
calculate(price)
show_price
End Sub
'
Function calculate(Byval number as Integer) as Integer
Return number + addition
End Function
'
Sub show_price()
print "the new price = " + calculate(price)
End Sub
```

Exercise

Rewrite the Sieve of Erathosthenes using a function to sieve.

True and False

Earlier we used already the notions true and false, for example in the program that tested the height of a price. We give now some advice how to make the program more readable when using the IF ... THEN statement. The outcome of such a statement is true or false. Bij declaring a boolean variable that descibes whether the outcome of a comparison is true or false we can write now simple code using such a variable. (Boolean variables are those that are used to decide between TRUE and FALSE). Look in the listings at the simple program “bools” how it can be done. Here part of it is shown:

```
number = val(letterSTR)
If number < 10 Then
 compareTF = True
Else
 compareTF = False
End If
If compareTF Then
 print letterSTR + "  < 10 "
Else
 print letterSTR + "  > 10 "
End If
```

It seems perhaps silly as we could have decided immediately in the first IF ... THEN statement what to do, but in large programs it makes the code more readable, especially with well choosen names for the variables. Note that we have written “compareTF” instead of “compare”. This is not obligatory but is done to make the code more readable, so now we know that compare is a boolean variable.

George Boole, developed an algebra with the following rules:

$1 + 0 = 1 \qquad 0 * 0 = 0$

$0 + 1 = 1 \qquad 0 * 1 = 0$

$1 + 1 = 1 \qquad 1 * 1 = 1$

Note the difference between “standard” algebra and “boolean algebra”: In boolean algebra 1 + 1 = 1.

In computer science, also in programming, this algebra is widely used. Write instead of “+” OR and instead of “*” AND.

Another logical operator is “NOT”. It simply reverses the value of the logical variable. See below how it can be used in the above program:

```
If not compareTF Then
  print letterSTR + "  > 10"
Else
  print letterSTR + "  < 10"
End If
```

Arithmetic laws to be tested by a program in BASIC

Some interesting arithmetical laws follow.

Nikomachos Of Gerasa who lived around 100 AC noted the following rule when computing the third power of integers:

$1^3 = 1$

$2^3 = 3 + 5$

$3^3 = 7 + 9 + 11$

$4^3 = 13 + 15 + 17 + 19$

$5^3 = 21 + 23 + 25 + 27 + 29$

etcetera

Note that a number to the third power can always be written as a number of consecutive odd numbers where the number of odd numbers is equal to the number that has to be risen tot the third power.

Exercise

Write a program to test the above stated rule.

Here follows another way of determining the square of a number: If a number is bigger than 15 and it ends with “5” then take the digits preceding the 5, add 1 to them and multiply it with the digits preciding the 5 . Multipy the outcome by 100 and add 25 to the result.

For example: we will calculate 125^2

First take 12 as this preceeds the 5. Add 1 so now we have 13. Multipply 13 with 12 which gives 156; mulitplied by 100 gives 15600; after adding 25 to this the result is 15625.

In this program we cannot use the function “val” to convert a string to a number as the string will contain more than one digit. In such a case “Cint can be used (convert integer):

```
input "Please, enter a number", letterSTR
number = Cint(letterSTR)
```

Write a program to test the rule.

We write now a number as getal 5. So 125 can be written as 12 5 and we call 12 “getfirst”

In a formula:

$\text{getal}^2 = (\text{getfirst} + 1) * \text{getfirst} * 100 + 25$

Solution to the first problem. We write first a formula.
Suppose we want to calculate 6 to the third power. The rule states the outcome is an additition of six odd numbers. We must now state were these six odd numbers start. It is the number that follows after the first 1 + 2 + 3 + 4 + 5 = 15 odd numbers. As there is always an even number in between we know now that it is the odd number following 15 * 2 = 30. So we start adding 6 odd numbers starting at

30 + 1 = 31. Then the outcome is 31 + 33 + 35 + 37 + 39 + 41 = 216

Write now the program to test this rule and then look at “nikom”.

The solution to the second problem is stored as “calcit”.

Introduction to forms

Until now we have only paid attention to the basics of Kbasic and its procedural aspects. Now it is time to be introduced to the instructions of Kbasic that make it possible to use “forms”, your own windows, and its event driven aspects. To do so we will make some of the programs developed in part one event driven. What characterisis Kbasic?

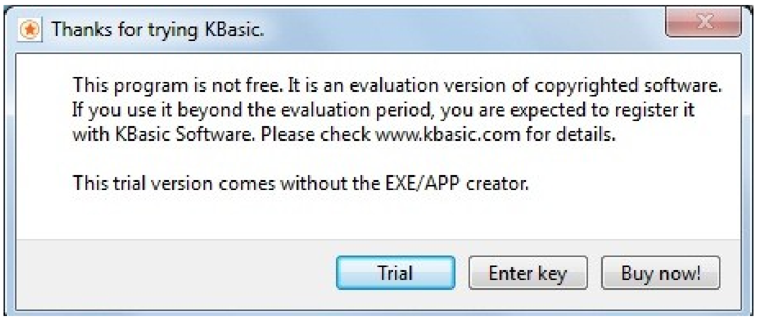
Kbasic uses windows, also called Forms.

Kbasic programs are event - driven.

Earlier the way a program is developed was called the “top down approach” and the way of ordering the subroutines “structured programming”. We used “Files” to store and execute our code. From now on we will still write structured code but the way a program is functioning when it is a “Project” is not top down but “event driven”. It means the program's execution depends on the actions taken by the user, such as clicking the mouse or using the keyboard. So the next step is to restructure the programs written in part one. First we determine what actions the user is allowed to take. Remember once your application is running he sees only your form as a window. Some of these actions are: clicking the buttons and entering text.

For those who skipped the first part of this tutorial because they think they have sufficient knwledge of the language we show which steps to take before inserting code.

When starting basic the following screen is displayed;


Thanks for trying KBasic.

This program is not free. It is an evaluation version of copyrighted software. If you use it beyond the evaluation period, you are expected to register it with KBasic Software. Please check www.kbasic.com for details.

This trial version comes without the EXE/APP creator.

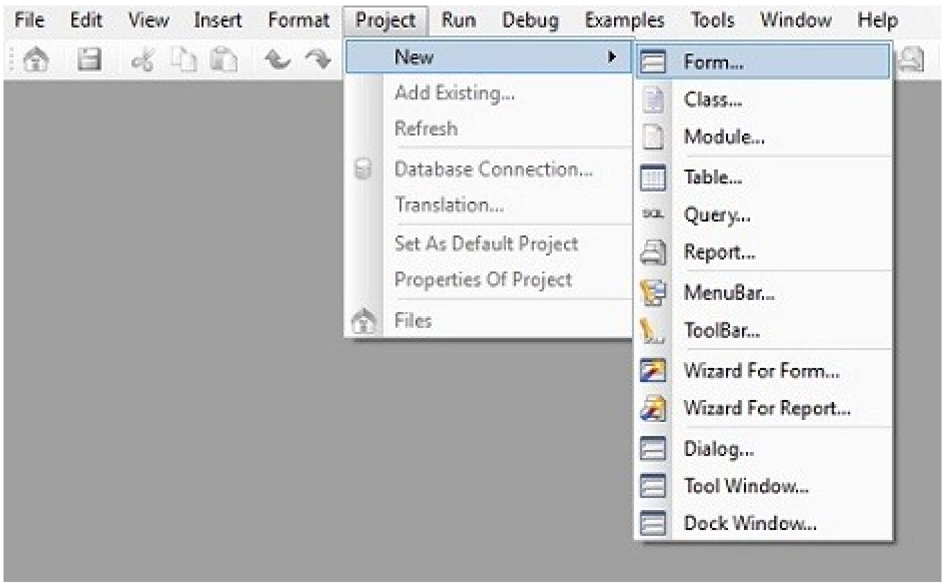
Depending on the actions you took earlier you click the appropriate button.

The nex screen is displayed:

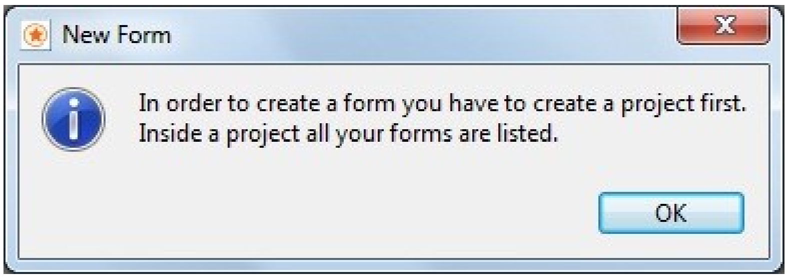
KBasic - NOT REGISTERED
File Edit View Insert Format Project Run Debug Examples Tools Window Help

An overwhelming amount of menu itmes and icons. They will be
explained as we proceed with the tutorial.

We open a new project, this time we will choose "Project – New Form" as shown below.

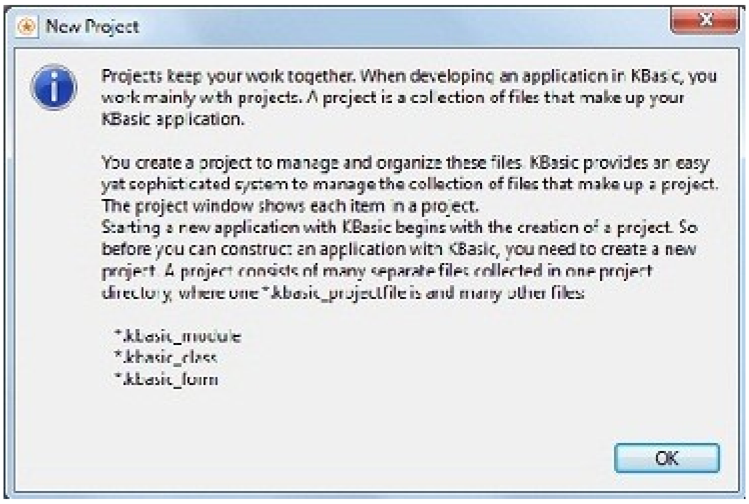


After clicking "Form" the next window opens:



In order to create a form you have to create a project first.
Inside a project all your forms are listed.

As has been explained in the firt part of this tutorial you must get accustomed to the fact that Kbasic will accompany your actions with providing messages considered worthwile. So when you click "OK" the next message is:

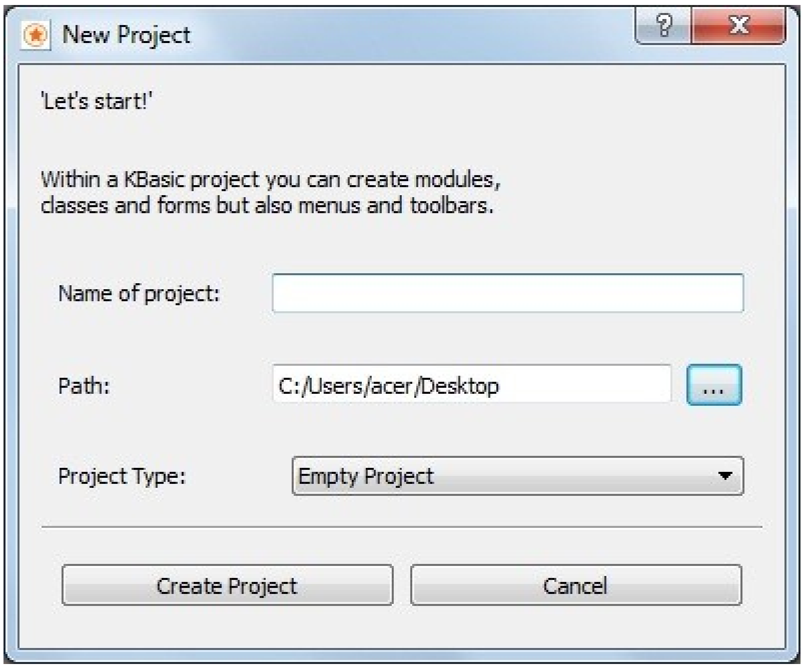


Projects keep your work together. When developing an application in KBasic, you work mainly with projects. A project is a collection of files that make up your KBasic application.

You create a project to manage and organize these files. KBasic provides an easy yet sophisticated system to manage the collection of files that make up a project. The project window shows each item in a project.
Starting a new application with KBasic begins with the creation of a project. So before you can construct an application with KBasic, you need to create a new project. A project consists of many separate files collected in one project directory, where one *.kbasic_projectfile is and many other files:

*.kbasic_module
*.kbasic_class
*.kbasic_form

Interesting, but not yet enough as the next message is:



'Let's start!'

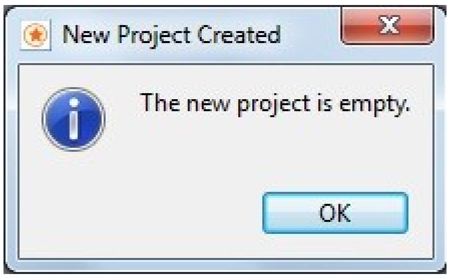
Within a KBasic project you can create modules, classes and forms but also menus and toolbars.

Name of project:

Path: C:/Users/acer/Desktop

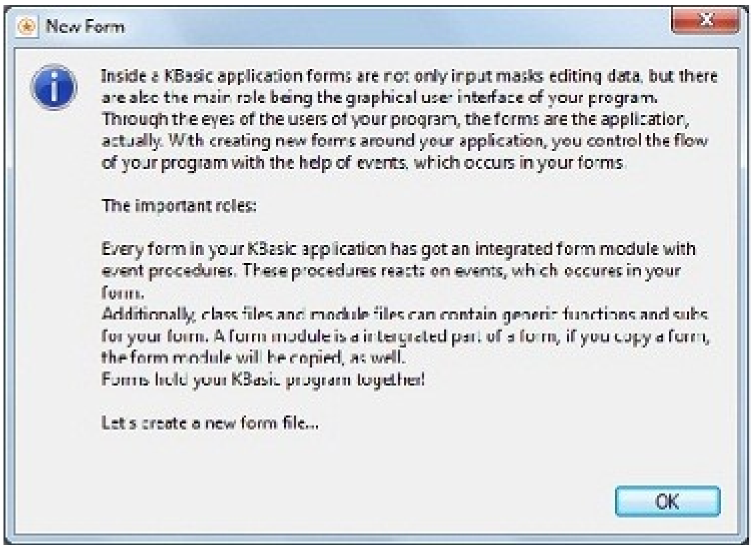
Project Type: Empty Project

Indeed, let's start now so we give the project a name, if considered necessary we change the path by clicking the button with the three blue dots. The project type we leave as proposed: “Empty Project”. This done we click “Create Project. So this information is stored but we again get a message:



The new project is empty.

Yes, we know and did it purposely. So we click “OK” and we are confronted with:



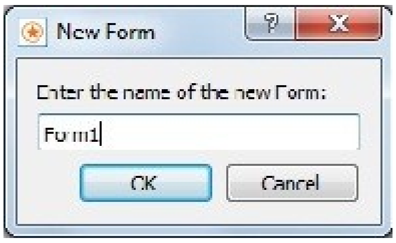
Inside a KBasic application forms are not only input masks editing data, but there are also the main role being the graphical user interface of your program. Through the eyes of the users of your program, the forms are the application, actually. With creating new forms around your application, you control the flow of your program with the help of events, which occurs in your forms.

The important roles:

Every form in your KBasic application has got an integrated form module with event procedures. These procedures reacts on events, which occures in your form.
Additionally, class files and module files can contain generic functions and subs for your form. A form module is a integrated part of a form, if you copy a form, the form module will be copied, as well.
Forms hold your KBasic program together!

Let's create a new form file...

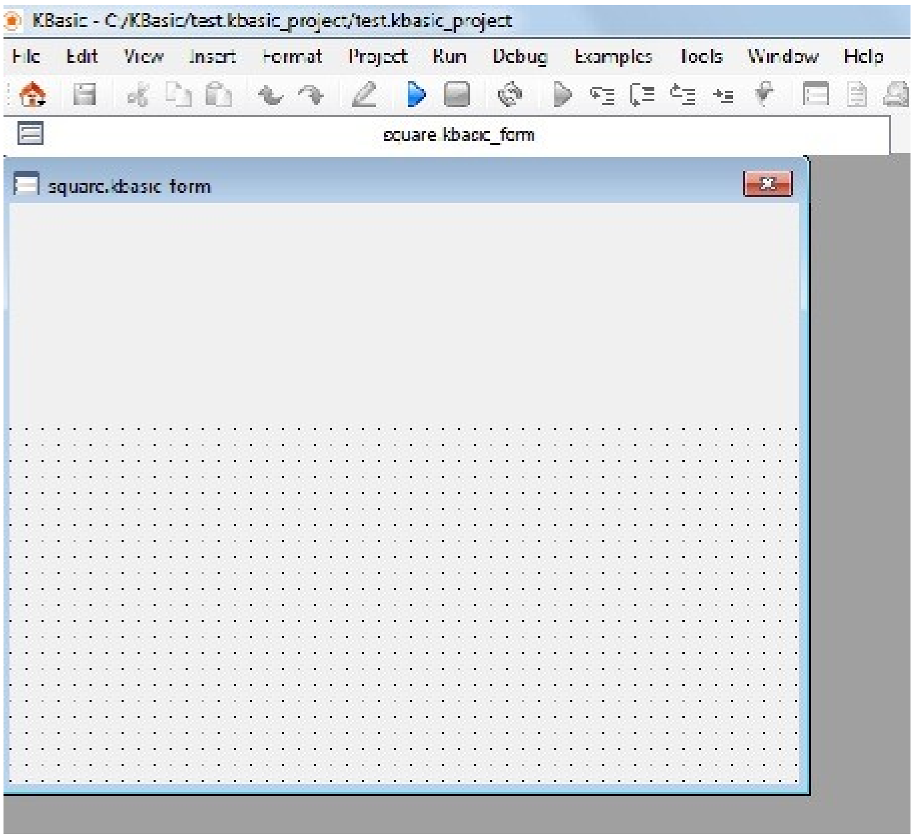
Yes, we again agree, let's create a new form file, so we click “OK” and the next window opens:



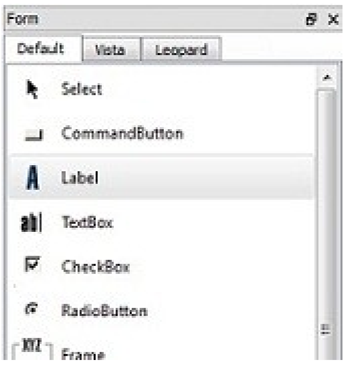
Enter the name of the new Form:

Form1

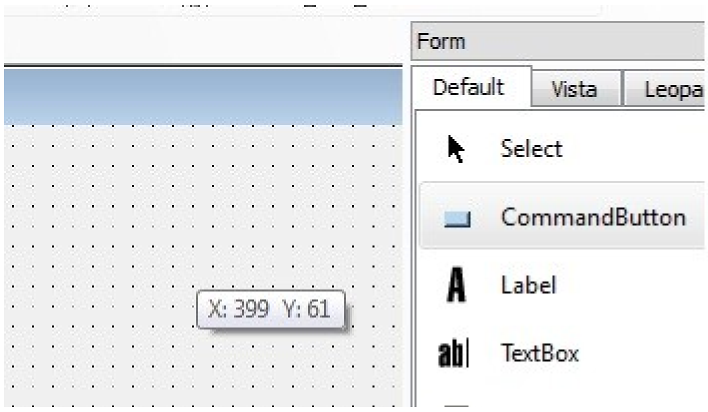
To be able to run the program described here it is obligatory to enter as name “Form1”. The name will displayed at the top of the new Form. Click “OK”.


KBasic - C:/KBasic/test.kbasic_project/test.kbasic_project
File Edit View Insert Format Project Run Debug Examples Tools Window Help
square.kbasic_form
square.kbasic_form

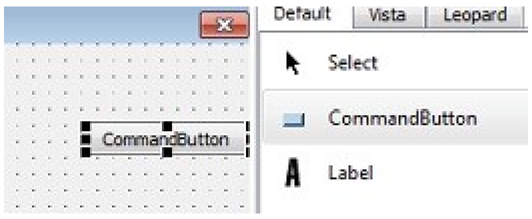
And here it is the window with the form we can work upon. We will put some items on this form. To get a list of these items we click once on the form and the list appears:



We will put two command buttons on the screen, one to display the square and the other to exit the application. Furthermore we will create room for the square by clicking the item “picture”. First we put a button on the form by moving the mouse to “CommandButton” and left click once. (If you click twice you get a message telling you what to do) The text “CommandButton” will become blue. We move the mouse on the form somewhere an click again.

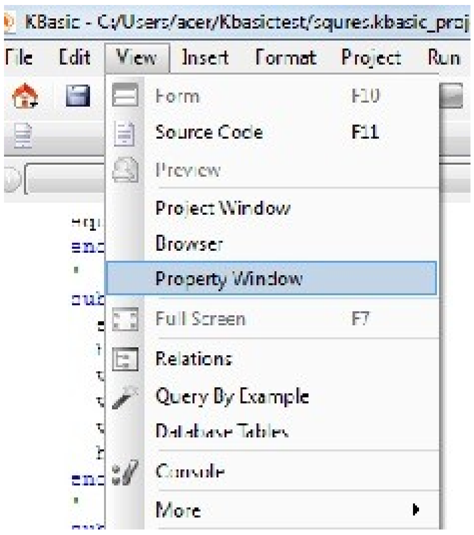


Now the button in its smallest size becomes visible:

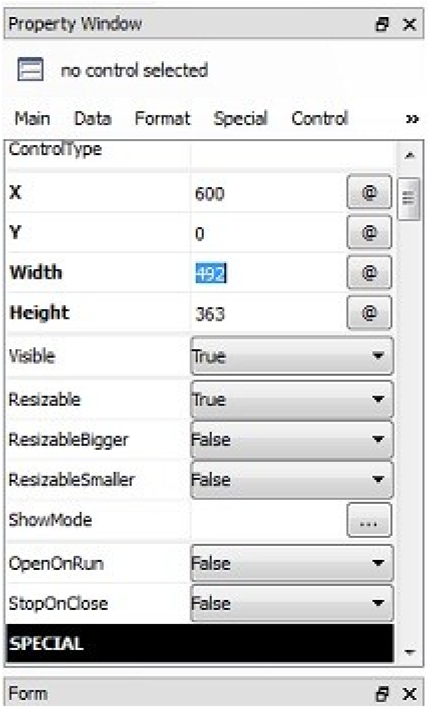


(If you made a mistake you can remove the item by right clicking the mouse and choose “Select all”. Then after you press the key “Del” the item will be deleted. If you want to delete more than one item, press delete repeatedly.

Now you can stretch this buttom in the customary way. After that we will change the text to “show square”. To do so we choose in the Menu “Property Window as shown below.



And at the right side of the window:


Property Window

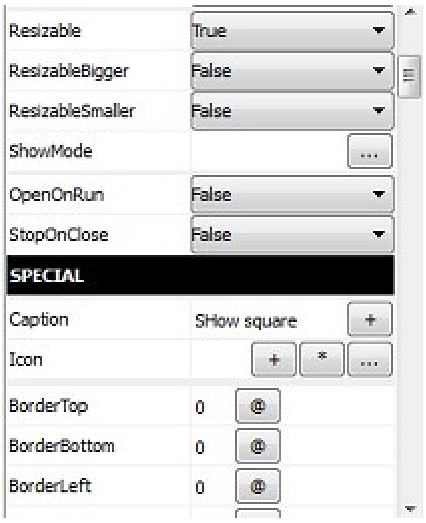
no control selected

Main Data Format Special Control »

ControlType	
X	600
Y	0
Width	492
Height	363
Visible	True
Resizable	True
ResizableBigger	False
ResizableSmaller	False
ShowMode	...
OpenOnRun	False
StopOnClose	False
SPECIAL	

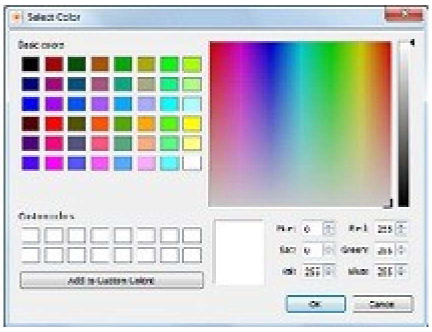
Form

we scroll “special” down to change “caption”:

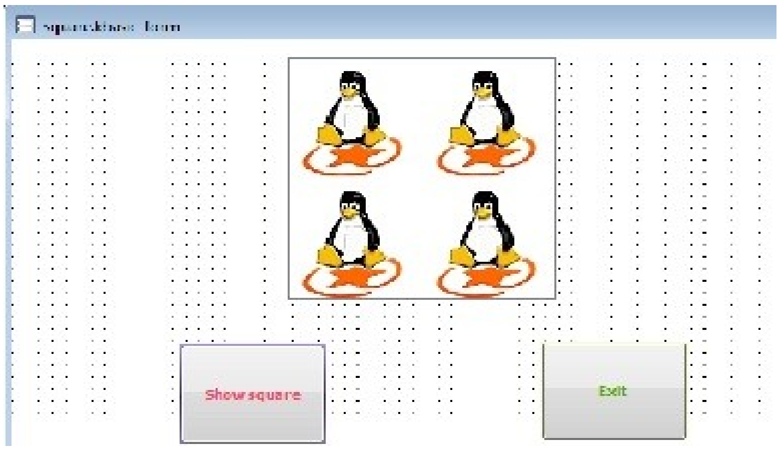


Resizable	True
ResizableBigger	False
ResizableSmaller	False
ShowMode	...
OpenOnRun	False
StopOnClose	False
SPECIAL	
Caption	SHow square
Icon	
BorderTop	0
BorderBottom	0
BorderLeft	0

to “.Show square”. There are more items we can give a value, but for the moment we only change the color of the button by scroloing further down until we find “font color”. If we click the following window opens enabling to choose a color:



We can do the same for the background. After inserting an “ImageBox” our form is now:



very funny but we want something else.

When clicking “View” there are two items we can choose: “Form” and “SourceCode”. You can choose whether you want to work on the Form or the program code by clicking the appropriate one. We chose “SourceCode” and type:

```
Private Sub CommandButton1_OnEvent()
  ImageBox1.Background = "c:\$AVG\001.jpg"
End Sub
Private Sub CommandButton2_OnEvent()
   end
End Sub
```

If you have chosen View – Form you can inspect and change if necessary the properties of your application. Be sure that the Name of the Imagebox is indeed “ImageBox1”, that the names of the buttons are “CommandButton1” and “CommandButton2” and that the picture mentioned between the quotes exixts. Earlier the “Property Window” has been discussed already. The result could be as shown below. I am sure you can produce something better.

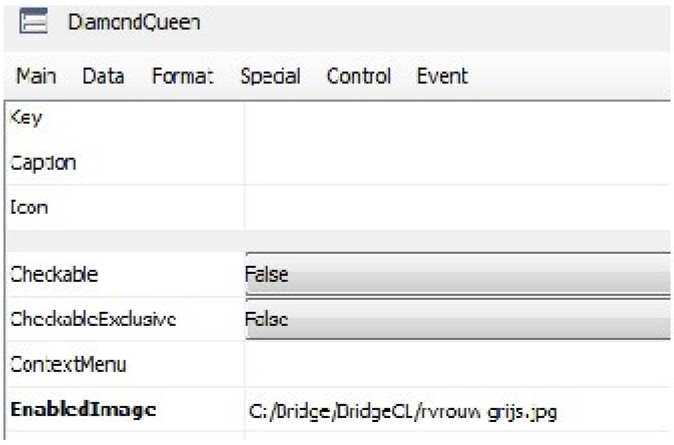
The next application will show a few more instructions to manipulate images. We start with a form containing three buttons which are: movebtn, showbtn and exitbtn. When the form is ready we load the picture by choosing the property window and then typing after “Enabled Image” where the picture is located. See the picture below.
As an example of the code involved look at the routine below:

```
Private Sub DiamondQueen_OnEvent()
DiamondQueen.X = 300
DiamondQueen.Y = 200
End Sub
```

The line “DiamondQueen.X = 300” is pronounced as “DiamondQueen her

X is three hundred". The character "X" following the dot described the X-coordinate, where the queen will be placed. Has it been for example the SpadesKing then the dot would have been pronounced as "his".

The picture of the queen has to inserted by typing after "Enabled Image" the location as shown below:

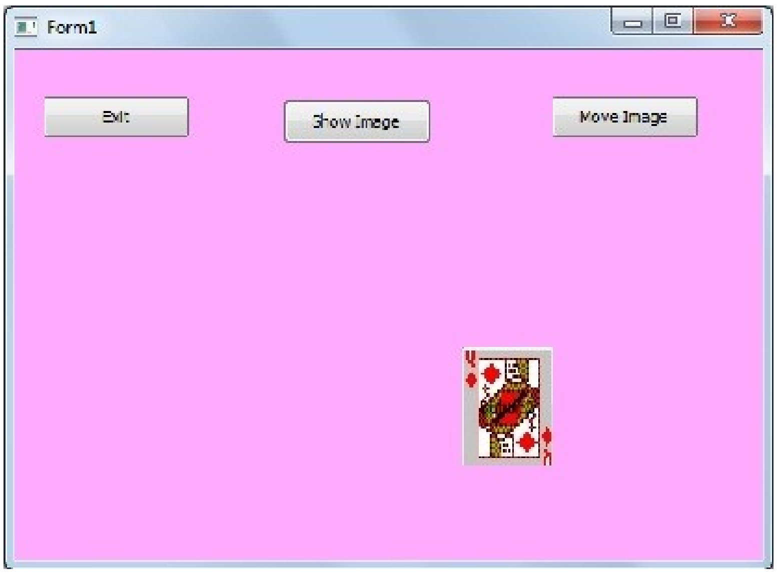


DiamondQueen

Main Data Format Special Control Event

Key	
Caption	
Icon	
Checkable	False
CheckableExclusive	False
ContextMenu	
EnabledImage	C:/Bridge/BridgeCL/vrouw grijs.jpg

After that has been done the form should look like:



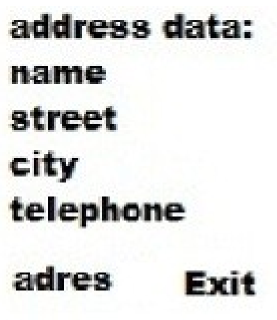
Here follows the complete listing of this small demo:

```
Sub showbtn_OnEvent()
'show image
DiamondQueen.visible=true
End Sub
'
Private Sub DiamondQueen_OnEvent()
DiamondQueen.X = 300
DiamondQueen.Y = 200
End Sub
'
Private Sub movebtn_OnEvent()
'move image
DiamondQueen.X = 400
DiamondQueen.Y = 150
End Sub
'
Private Sub exitbtn_OnEvent()
End
End Sub
```

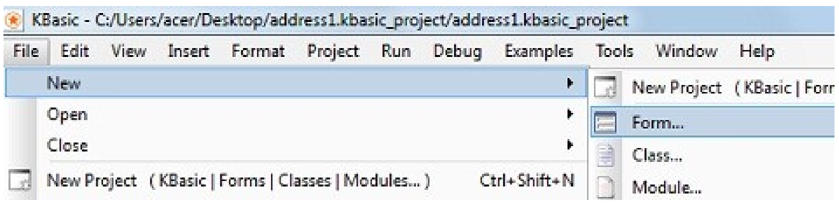
As is clear form the code provided you can move the picture in this demo by clicking a button or the picture. It depends on the application what to do with a picture.

Experiment with the code for picture manipulation to get accustomed to it.

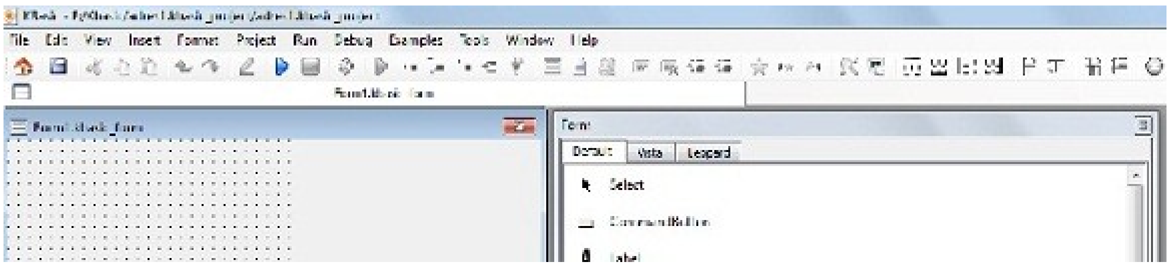
Our next application will be the “adres” one. By developing it we will learn about a few other control items. First we design our window:



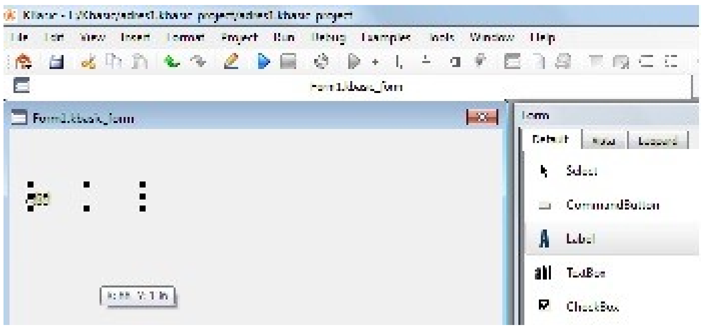
Now that the desing is ready we implement it in Kbasic. To do so create a new project. As we anyhow willl use a “form” we click “Form” as shown below.


KBasic - C:/Users/acer/Desktop/address1.kbasic_project/address1.kbasic_project
File
Edit
View
Insert
Format
Project
Run
Debug
Examples
Tools
Window
Help
New
Open
Close
New Project (KBasic | Forms | Classes | Modules...) Ctrl+Shift+N
New Project (KBasic | Forr
Form...
Class...
Module...

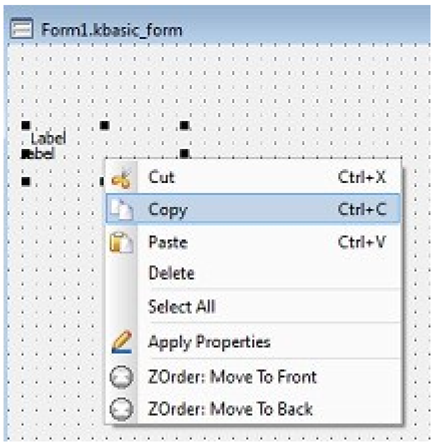
We will get a message telling us to creat a project first. But after you click OK you can proceed in the usual way, dont forget to enter as name for the form “Form1”. That form will be visible together with the « toolbox « as shown below:



We click the item « Label » as it will be the first item we place on our form. When the toolbox covers the form just move it to the right. Then press the left button of the mouse at a convenient place of the form and release the mouse. If this position is not the right one you can later adjust it. The form should now look as:



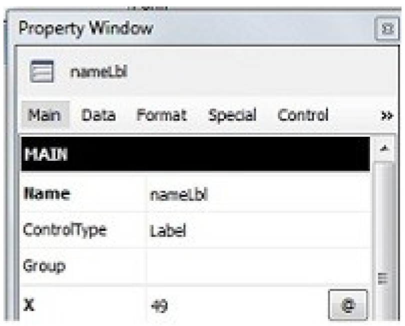
Now place the cursor on « Label1 » and click the **right** mouse button and click “Copy” as shown below:



After you clicked “Copy” do the same for “Paste” as many times as you need labels. For this application we need four: name, street, city and telephone.

We click the menu item “View” and then chose “Property Window”. The items we change will be discussed now. If the slider is not in its upmost position move it upwards so the first items become visible.

We will now adjust the first label under the condition that we had chosen it. When you click in the form another item the property window changes accordingly. In “Special” we change the caption. It is good practice to end the name with « lbl » so we are sure when inserting code it is a label we are handling.


Property Window

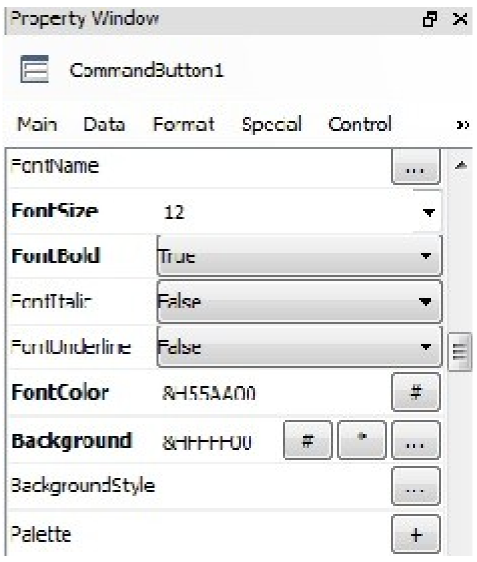
nameLbl

Main Data Format Special Control »

MAIN

Name	nameLbl
ControlType	Label
Group	
X	49

If we click the small "+" sign we can see the content in a separate window. We move down in the "Property window" to "Property (Control)" and change the aspect of the label. Here are set Fontname: courier New, Fontsize 16, Bold true, Fontcolor and Background color.

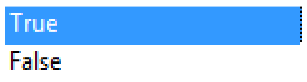

Property Window

CommandButton1

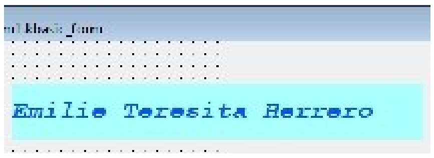
Main Data Format Special Control »

FontName	
FontSize	12
FontBold	True
FontItalic	False
FontUnderline	False
FontColor	&H55AA00
Background	&HFFFF00
BackgroundStyle	
Palette	

You only need to type the font size. Fontname you can choose from a list.
Bold, italic and underscore you set to true or false by clicking:



The result for this label is:

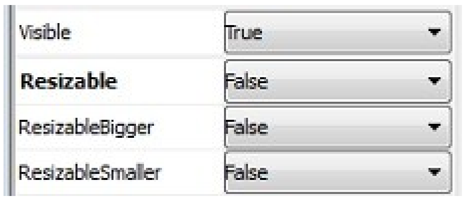


Do the same for the other labels: street, city, and telephone.

We discuss a few other items: In “Main” of the “Property window” “Visible is set to “false” for this labelbecause we want it to become visible after clicking the button “show”. This is done for all labels.

X, Y, Width and Heigt speak for itself, the x and y coordinates are shown. You can change the coordinates. If for example coordinate Y is changed. the label is moved upwards a little bit and will be shown immediately in the form.

If you don’t want the users of your program to tamper with the window of your appication, the items displayed below should be as:

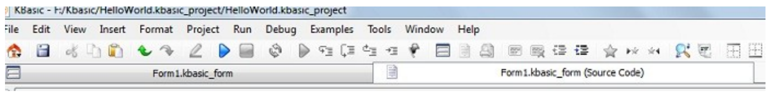


Visible	True
Resizable	False
ResizableBigger	False
ResizableSmaller	False

Visible must remain true of course.
The butttons should be visible otherwise the user can not invoke an action.

We have aleady prepared a nice button image with our paint program that you can use in your applications

We are finished with the form part of our program, so now the code has to be entered. We click on the form outside he labels and button or we press the key “F11” or we chose menu item View and chose “Source code”, or lastly we click the approriate left or right item as shown below, so possibilities enough.



KBasic - F:/Kbasic/HelloWorld.kbasic_project/HelloWorld.kbasic_project

```
Private Sub CommandButton1_OnEvent()
  nameLbl.visible=true
  streetLbl.visible=true
  cityLbl.visible=true
  phoneLbl.visible=true
End Sub
'
Private Sub CommandButton2_OnEvent()
 end
End Sub
```

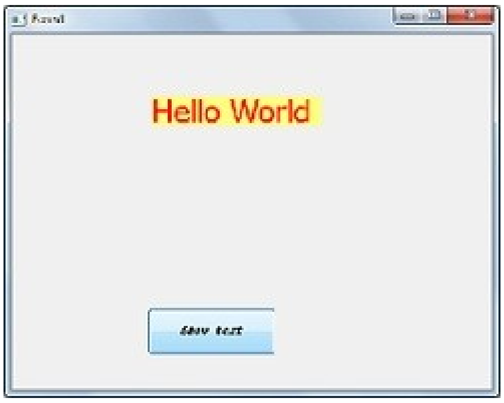
Actually we need only to enter the code between “sub” and “end sub” as when you click the buttons on your form sucessivly Kbasic will prepare

the event-subroutine for you.
It is a good idea to save our work this far by clicking “save all” in the File menu. Always close a project first before opening another to keep your workspace tidy!

Run the program and admire your Kbasic program.

Editing your program

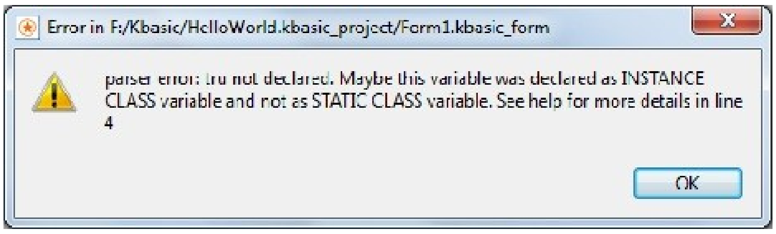
Let us practise with a small program. Every serious course about programming should discuss the « hello world » program. So below it is, but unfortunatelyit contains an error. It needs improvement. Please copy it to a clean slab. The correct program should show:



Prepare this application and click the button “show text” to enter code between “sub” and “end sub”. Unfortunately the text is erronous:

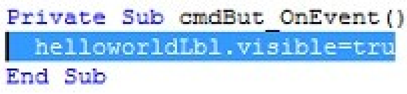
helloworldLbl.visible=tru

Enter with the error to see the result.


Error in F:/Kbasic/HelloWorld.kbasic_project/Form1.kbasic_form

parser error: tru not declared. Maybe this variable was declared as INSTANCE CLASS variable and not as STATIC CLASS variable. See help for more details in line 4

Furthermore the line containing the error is highlighted:

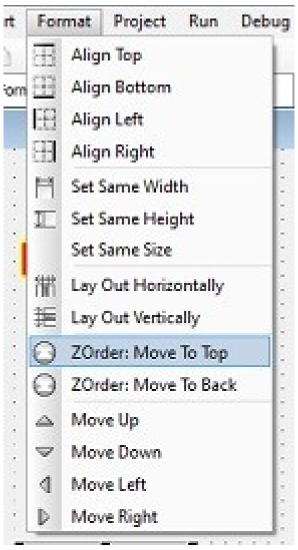

```
Private Sub cmdBut_OnEvent()
  helloworldLbl.visible=tru
End Sub
```

The error message says “tru” is not declared. It was not our intention to introduce a variable of that name, so the error is in “tru”. It should be written as “true”.

Correcting an item on the form

If you want to make corrections to an item on the form you click it once and the item will be selected. Clicking it twice will show its code.

Suppose you want to look at the item but it is hidden behind another one. You can do it easily as indicated below:

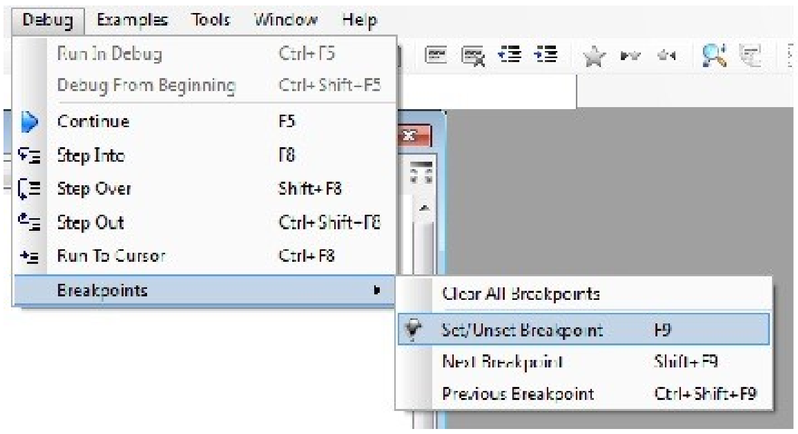


You can click bring to back the item not wanted until the one you want to look at appears.

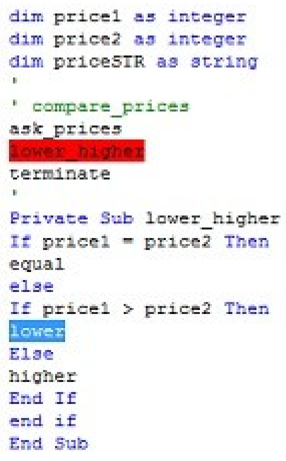
Debugging

Look at the example "rude". When you run this code you will notice that it does not function as intended. So something is wrongly coded. In a simple program as this it is not difficult to find what is wrong, but it is used to explain how to debug in Kbasic. Try to find the bug now by running the program, but do not inspect the code as we will detect the bug by using the debugger**.** So take a look and copy the program "rude" as "newfile".

When entering 5 and 7 for example, the answer should be that 5 is lower than 7. Perhaps there is something wrong with the code of the subroutine "lower_higher" so we wil insert a breakpoint at the instruction of its call. To do so we move the crusor to that line and click key "F9" or follow whas been shown below:

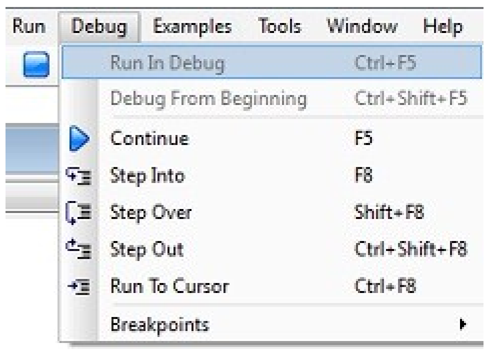


after that the breakpoint is indicated as follows:

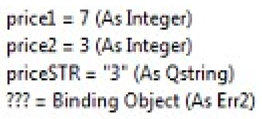


```
dim price1 as integer
dim price2 as integer
dim priceSTR as string
'
' compare_prices
ask_prices
lower_higher
terminate
'
Private Sub lower_higher
If price1 = price2 Then
equal
else
If price1 > price2 Then
lower
Else
higher
End If
end if
End Sub
```

Then start the program by clicking "Run in Debug" as shown below:



When the breakpointis reached information will be displayed at the bottom of the sreen, when "7" and "3" were typed a s hown:



```
price1 = 7 (As Integer)
price2 = 3 (As Integer)
priceSTR = "3" (As Qstring)
??? = Binding Object (As Err2)
```

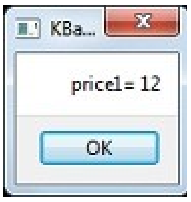
We continue single-stepping by pressing key "F8". Now it becomes clear that the wrong subroutine was called. The subroutines "lower" and "higher" should be exchanged or the sign ">" should be reversed.

Note that usually you cannot correct code as long as the program is running. It is therefore essential to stop the program.

Sometimes it is inconvenient to use the debug option as it is not possible to debug and at the same time see on the screen what happens with the form. In such a case it might he handy to insert a message instruction. See the piece of code below:

MsgBox("price1= " + price1)

When the line containing this instruction is executed the result is:



price1= 12

under the assumption that “12” was typed as price1.

As has been said you can continue debugging by clicking the F8 key. Unfortunately that does not function when another program, for example a text-checquer is running and uses the same key.

Save what you have done so far by clicking “**Save all**”.

Suppose you are happy with the program you have written and want so send it to a friend, but you don't want to send the source code, only a version of the program he can run. In that case you have to “build” as it is called, code: an executable version of your program. To do so you open the menu item “debug” and choose the line “Build”. But be aware that this funtions only if you bought Kbasic,and registered, of course!

Listings K7Basic:

square1:

```
square

end
'
sub square
horizontal
vertical
vertical
vertical
horizontal
end sub
'
sub horizontal
print "---------"
end sub
'
sub vertical
print "|  |"
end sub
'_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

emptyline:

```
' the subroutine empty line draws
'an empty line
sub emptyline
print " "
end sub
'_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

adres1:

```
' this program prints an address
dim personSTR as string
dim streetSTR as string
dim citySTR as string
dim landline as integer
address
terminate
End
'
sub address
person
```

```
city
street
telephone
showAddress
end sub
'
sub person
personSTR="Emilie T. Sagario"
end sub
'
sub street
streetSTR="Adamville Compound"
end sub
'
sub city
citySTR="Marigondon Lapulapu City"
end sub
'
sub telephone
landline = 95478476
end sub
'
sub showAddress
print personSTR
print streetSTR
print citySTR
print "tel " +landline
end sub
'
sub terminate
print " "
print "done"
end sub
'
___________________
```

price1:

```
Dim price As Short
Const addition As Short = 2
Dim priceSTR As String
'
newprice
end
'
Sub newprice
ask_price
calculate
show_price
End Sub
'
Sub calculate
price = price + addition
End Sub
'
sub ASK_PRICE
input "Enter a price"; priceSTR
price=val(priceSTR)
end sub
'
Sub show_price()
print "the new price = " + price
End Sub
'_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

price2:

```
dim price1 as integer
dim price2 as integer
dim priceSTR as string
'
compare_prices
end
'
sub compare_prices
ask_prices
lower_higher
terminate
end sub
'
```

```
Private Sub lower_higher
If price1 < price2 Then
lower
Else
higher
End If
End Sub
'
Private Sub lower
print "price 1 is lower than price 2"
End Sub
'
Private Sub higher
print "price 1 is higher than price 2"
End Sub
'
Sub terminate
print ""
input "Enter a character to stop";priceSTR
price1 = val(priceSTR)
End sub
'
sub ask_prices
input "Enter a price";priceSTR
price1 = val(priceSTR)
input "Enter another price";priceSTR
price2 = val(priceSTR)
end sub
'
____________________
```

price3:

```
dim price as integer
dim priceSTR as string
'
determine_prices
end
'
sub determine_prices
ask_price
calculate
terminate
end sub
Private Sub calculate
If price <10 Then
higher1
else
If price > 25 Then
higher5
Else
higher2
End If
end if
End Sub
'
Private Sub higher1
price = price + 1
End Sub
'
Private Sub higher5
price = price +5
End Sub
'
Sub higher2
price = price +2
End Sub
'
Sub terminate
print "The new price is: " + price
print ""
input "Enter a character to stop";priceSTR
price = val(priceSTR)
End sub
'
sub ask_price
input "Enter a price";priceSTR
price = val(priceSTR)
end sub
'
____________________
```

manyifs:

```
Dim price As integer

Dim priceSTR As String
much_ifs
end
'
Sub much_ifs
' this program accepts a price between 0 and 10
Convert
End
End Sub
'
Sub Convert
What_Price
Determine
End Sub
'
Sub What_Price
input "Please, enter a price between 0 and 10"; priceSTR
price = val(priceSTR)
End Sub
'
Sub Determine
if price = 0 then
print "I said between zero and ten!"
else
if price = 1 then
print "The price is one"
else
if price = 2 then
print "The price is two"
else
if price = 3 then
print "The price is three"
else
if price = 4 then
print "The price is four"
else
if price = 5 then
print "The price is five"
else
if price = 6 then
print "The price is six"
else
if price = 7 then
print "The price is seven"
else
if price = 8 then
print "The price is eight"
else
if price = 9 then
print "The price is nine"
Else
print "The price must be < 10"
End if
End if
End if
End if
End if
End if
End if
End if
End if
End if
End Sub
'
'
____________________
```

whichar:

```
dim value as short = 12
dim letterSTR as String
dim new_letterSTR as String
'
convert
print""
print "done"
end
'
```

```
Sub convert
Ask_char
Determine()
End Sub
'
Private Sub Determine
If letterSTR <> "0" Then
value = Asc(letterSTR)
value= value-96
new_letterSTR = Chr(value + 1)
Print "This is letter " + value + " of the alfabet"
Else
Print "this was a zero so the program ended"
end
End If
End Sub
'
sub ask_char
input "Enter a character"; letterSTR
end sub
```

_ _

nexchar:

```
dim value as short = 12
dim letterSTR as String
dim new_letterSTR as String
'
convert
print""
print "done"
end
'
Sub convert
Ask_char
Determine()
End Sub
'
Private Sub Determine
If letterSTR <> "0" Then
value = Asc(letterSTR)
new_letterSTR = Chr(value + 1)
Print "The next letter of the alfabet is: " + new_letterSTR
Else
Print "this was a zero so the program ended"
end
End If
End Sub
'
sub ask_char
input "Enter a character"; letterSTR
end sub
'
```

_ _

charfuncs

```
Dim str As String = "The quick brown fox jumps over the lazy dog."

Dim str2 As String = str
Dim result As String
result = left(str, 19)
print result
result = right(str, 4)
print result
result = Mid(str, 5, 15)
print result
result = Mid(str, 1, 30)
print result
'Mid$ in some old version of
'E.G. Mid$(str,41,3)="cat" would replace dog with cat.
'The string.Replace is okay for a single instance of a word.>>
str = str.Replace("dog", "cat")
print str
'It is NOT okay where two of the same word exist.>>
str = str.Replace("cat", "the")
'Now we have two of the word "the"
str = str.Replace("the", "very")
print str
'Reset original string "str" to the original string.
str = str2
print str
'Same as Mid$(str,5,5)="smart"
'The 2nd "5" is not needed as the FUNCTION uses the String.Length
str = Replace(str, 5, "smart")
```

```
print str
```

_ _

while:

```
dim value as integer

dim new_letterSTR as String
dim letterSTR as String
'
convert
end
'
Private Sub convert
While letterSTR <> "0"
Ask_char
Determine()
End While
End Sub
'
sub determine
value = Asc(letterSTR)
new_letterSTR = Chr(value + 1)
Print "The next letter of the alfabet is: "+ new_letterSTR
end sub
'
sub ask_char
input "Enter a character"; letterSTR
end sub
```

_ _

chnot

```
dim value as short = 5
dim letterSTR as String
dim new_letterSTR as String
'
convert
print""
print "done"
end
'
Private Sub convert
Do Until value = 0
ask_char
Determine
Loop
End Sub
'
Private Sub Determine
If letterSTR <> "0" Then
value = Asc(letterSTR)
new_letterSTR = Chr(value + 1)
Print "The next letter of the alfabet = " + new_letterSTR
Else
Print "this was a zero so the program ended"
end
End If
End Sub
'
sub ask_char
input "Enter a character"; letterSTR
end sub
```

_ _

pricewithifs

```
dim price as short = 5

dim priceSTR as string
Convert()
End
'
Sub Convert()
do until price = 0
What_Price()
Determine()
Loop
Print "That is it, done"
End Sub
'
```

```
Sub What_Price()
Input "Please, enter a price between 0 and 10"; priceSTR
price=val(priceSTR)
End Sub
'
Private Sub Determine()
If price = 0 Then
Print "this was a zero so the program ended...."
'End
Else
If price = 1 Then
print "The price is one"
Else
If price = 2 Then
print "The price is two"
Else
If price = 3 Then
print "The price is three"
Else
If price = 4 Then
print "The price is four"
Else
If price = 5 Then
print "The price is five"
Else
If price = 6 Then
print "The price is six"
Else
If price = 7 Then
print "The price is seven"
Else
If price = 8 Then
print "The price is eight"
Else
If price = 9 Then
print "The price is nine"
Else
If price > 9 Then print "The price should be one digit only!!!!!"
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

testchar with value=0

```
dim value as short = 0

dim new_letterSTR as String
dim letterSTR as String
'
convert
'
Sub convert
Do Until value = 0
Ask_char
Determine()
Loop
End Sub
'
Private Sub Determine
If letterSTR <> "0" Then
value = Asc(letterSTR)
new_letterSTR = Chr(value + 1)
Print "The next letter of the alfabet is: " + new_letterSTR
Else
Print "this was a zero so the program ended"
end
End If
End Sub
'
sub ask_char
input "Enter a character"; letterSTR
end sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

selcas

```
Dim price As Short = 8

dim priceSTR as string
Dim a As String
' this program accepts a price between 0 and 10 Dim price As Short
Convert
End
'
Sub Convert
What_Price
Determine
End Sub
'
Sub What_Price
input "Please, enter a price between 0 and 10";priceSTR
price = val(priceSTR)
End Sub
'
Sub Determine
Select Case price
Case 0
print "I said between zero and ten!"
Case 1
print "The price is one"
Case 2
print "The price is two"
Case 3
print "The price is three"
Case 4
print "The price is four"
Case 5
print "The price is five"
Case 6
print "The price is six"
Case 7
print "The price is seven"
Case 8
print "The price is eight"
Case 9
print "The price is nine"
Case Else
print "The price must be < 10"
End Select ' price

End Sub
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

tennum

```
Dim I, max, number[10] As Short

Dim cipherSTR As String
'
maximum
'
Sub maximum
getnumbers
decide
End Sub
'
Sub getnumbers
Dim cipher As Short
For I = 1 To 10
print "Enter number"
input "cipher = "; cipherSTR
cipher=val(cipherSTR)
number(I) = cipher
Next I
End Sub
'
Sub decide
Dim cipher, oldcipher, maximum As Short
oldcipher = 0
For I = 1 To 10
cipher = number[I]
If cipher > oldcipher Then
max = cipher
oldcipher = cipher
End If
Next I
```

```
print ""
print "The maximum is " + max
End Sub
```

____

funcsquare

```
' this program accepts a digit and prints its square

DIM letterSTR as string
DIM value as Integer
Convert()
End
'
Function square(ByVal n as Integer) as integer
Return n * n
End Function
'
Private Sub Convert()
Ask_number()
Determine()
End Sub
'
Private Sub Ask_number()
Input "Please, enter a number";letterSTR
End Sub
'
Private Sub Determine()
value = val(letterSTR)
print "the square is: " + square(value)
End Sub
```

____

muchifs

```
Dim price As integer

Dim priceSTR As String
much_ifs
end
'
Sub much_ifs
' this program accepts a price between 0 and 10
Convert
End
End Sub
'
Sub Convert
What_Price
Determine
End Sub
'
Sub What_Price
input "Please, enter a price between 0 and 10"; priceSTR
price = val(priceSTR)
End Sub
'
Sub Determine
if price = 0 then
print "I said between zero and ten!"
else
if price = 1 then
print "The price is one"
else
if price = 2 then
print "The price is two"
else
if price = 3 then
print "The price is three"
else
if price = 4 then
print "The price is four"
else
if price = 5 then
print "The price is five"
else
if price = 6 then
print "The price is six"
else
if price = 7 then
print "The price is seven"
```

```
else
if price = 8 then
print "The price is eight"
else
if price = 9 then
print "The price is nine"
Else
print "The price must be < 10"
End if
End if
End if
End if
End if
End if
End if
End if
End if
End if
End Sub
'
```

____

fadres1

```
Dim lineSTR As String

File.Create("D:\adres.txt")
OPEN "D:\adres.txt" FOR OUTPUT AS #1
lineSTR= " naam en adres "
print lineSTR
print #1,lineSTR
'print #1, "|"
input "name"; lineSTR
print lineSTR
print #1,lineSTR
print #1, "|"
input "Enter adres"; lineSTR
print lineSTR
print #1,lineSTR
print #1, "|"
input "Enter city"; lineSTR
print lineSTR
print #1,lineSTR
print #1, "|"
input "country"; lineSTR
print lineSTR
print #1,lineSTR
print #1, "|"
input "landline"; lineSTR
print lineSTR
print #1,lineSTR
print #1, "|"
print #1, " "
Print "done"
Close #1
end
```

____

squareswithoutfornext

```
Dim number As Short

compute
End
Sub compute
number = 2
Print "The square of two is ", number * number
number = 3
Print "The square of three is ", number * number
number = 4
Print "The square of four is ", number * number
number = 5
Print "The square of five is ", number * number
number = 6
Print "The square of six is ", number * number
number = 7
Print "The square of seven is ", number * number
number = 8
Print "The square of eight is ", number * number
number = 9
```

```
Print "The square of nine is ", number * number
number = 10
Print "The square of ten is ", number * number
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

fibonacci

```
Dim month as integer = 0
dim youngrabbits, grownuprabbits, notyet As integer
Dim previous as integer
Dim a As String
'
Fibonacci
End
'
Sub Fibonacci
youngrabbits = 1
Do Until month = 13
increment
Loop
print " "
print "This ends the Fibonacci program"
End Sub
'
Sub increment
month = month + 1
previous = month - 1
If month > 1 Then
Print "in month " + previous + " there are "+ grownuprabbits+ "rabbits"
End If
grownuprabbits = youngrabbits + notyet
notyet = youngrabbits
youngrabbits = grownuprabbits
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

erathos

```
'the sieve of Erathosthenes

'this program sieves the numbers 1 to 100 so only primes remain.
Dim number[100]
DIM I as integer
'
Eratos
print ""
print "this ends the sieve program"
End
'
Sub Eratos
fillarray
sieve
showresult
End Sub
'
Sub fillarray
For I = 1 To 100
number[I] = I
Next I
End Sub
'
Sub sieve
Dim thisnumber, rest, I As Short
For I = 1 To 100
thisnumber = number[I]
rest = thisnumber Mod 2
If rest = 0 Then
number[I] = 0
End If
Next I
For I = 1 To 100
thisnumber = number[I]
rest = thisnumber Mod 3
If rest = 0 Then
number[I] = 0
End If
Next I
For I = 1 To 100
thisnumber = number[I]
rest = thisnumber Mod 5
```

```
If rest = 0 Then
number[I] = 0
End If
Next I
For I = 1 To 100
thisnumber = number[I]
rest = thisnumber Mod 7
If rest = 0 Then
number[I] = 0
End If
Next I
End Sub
'
Sub showresult
Dim thisnumber As Short
For I = 1 To 100
thisnumber = number[I]
If thisnumber <> 0 Then
print "number " + thisnumber + " is a prime"
End If
Next I
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

compareprices:

```
dim price1 as integer
dim price2 as integer
dim priceSTR as string
'
compare_prices
end
'
sub compare_prices
ask_prices
lower_higher
terminate
end sub
Private Sub lower_higher
If price1 < price2 Then
lower
Else
higher
End If
End Sub
'
Private Sub lower
print "price 1 is lower than price 2"
End Sub
'
Private Sub higher
print "price 1 is higher than price 2"
End Sub
Sub terminate
print ""
input "Enter a character to stop";priceSTR
price1 = val(priceSTR)
End sub
sub ask_prices
input "Enter a price";priceSTR
price1 = val(priceSTR)
input "Enter another price";priceSTR
price2 = val(priceSTR)
end sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

bools:

```
Dim value As Short

Dim letterSTR As String
' this program accepts a number and prints some information
convert()
End
'
Private Sub convert()
Do Until letterSTR = "0"
Ask_Number()
Determine()
```

```
Loop
End Sub
'
Private Sub Ask_Number()
input "Please, enter a number ", letterSTR
End Sub
'
Private Sub Determine()
Dim number As Short
Dim compareTF As Boolean
number = val(letterSTR)
If number < 10 Then
compareTF = True
Else
compareTF = False
End If
If compareTF Then
print letterSTR + " < 10 "
Else
print letterSTR + " > 10 "
End If
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

price byval

```
Dim price As Integer

Dim newprice as Integer
Const addition as Integer = 2
'
addprice
end
'
Sub addprice
price = 25
calculate(price)
show_price
End Sub
'
Sub calculate(ByVal pretium)
newprice = pretium + addition
End Sub
'
Sub show_price()
print "the new price = " + newprice
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

rude

dim price1 as integer

```
dim price2 as integer
dim priceSTR as string
'
' compare_prices
ask_prices
lower_higher
terminate
'
Private Sub lower_higher
If price1 = price2 Then
equal
else
If price1 > price2 Then
lower
Else
higher
End If
end if
End Sub
'
Private Sub lower
print "price 1 is lower than price 2"
End Sub
'
Private Sub higher
print "price 1 is higher than price 2"
End Sub
```

```
'
Sub equal
print "price 1 is as high as price 2"
End Sub
'
Sub terminate
print ""
input "Enter a character to stop";priceSTR
price1 = val(priceSTR)
End sub
'
sub ask_prices
input "Enter a price";priceSTR
price1 = val(priceSTR)
input "Enter another price";priceSTR
price2 = val(priceSTR)
end sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

ascfunction

```
' this program accepts a letter and prints its ascii value

Dim letterSTR As String
Dim value as short
main
end
'
Sub Main
Ask_character
Determine
End Sub
'
Private Sub Ask_character
input "Please, enter a letter"; letterSTR
End Sub
'
Private Sub Determine
value = Asc(letterSTR)
print "The ascii value = "; value
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

price with const

```
Dim price As Short

Dim priceSTR As String
Const addition As Short = 2
'
newprice
end
'
Sub newprice
ask_price
calculate
show_price
End Sub
'
Sub calculate
price = price + addition
End Sub
'
sub ASK_PRICE
input "Enter a price"; priceSTR
price=val(priceSTR)
end sub
'
Sub show_price()
print "the new price = " + price
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

askprice

```
' this program adds 2 pesos to a price of 25
```

```
Dim addition, price As Short
Dim priceSTR As String
'
newprice
end
'
Sub newprice
ask_price
calculate
show_price
End Sub
'
Sub calculate
addition = 2
price = price + addition
End Sub
'
sub ASK_PRICE
input "Enter a price"; priceSTR
price=val(priceSTR)
end sub
'
Sub show_price()
print "the new price = " + price
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

nikom

```
'this program prints a number to the third power
Dim number As Short
Dim i As Short
Dim j As Short
Dim letterSTR As string
Dim counter As Short
Dim remain As Short
Dim result As Short
ThirdPower()
print "done"
end
'
Sub ThirdPower()
WhichNumber()
Calculate()
End Sub
'
Sub WhichNumber()
input "Please, enter a number between 0 and 10 ",letterSTR
End Sub
'
Sub Calculate()
number = (asc(letterSTR))-48
If number = 0 Then
Print "Zero to the third power is zero, of course"
Else
For i = 1 To number - 1
counter = counter + i
Next i
counter = (counter * 2) + 1
For j = counter To counter + ((number * 2) - 1)
remain = j Mod 2
If remain <> 0 Then
result = result + j
End If
Next j
End If
Print"the third power = ", result
End Sub
```

Note: Use counter as variable name and not count!

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

calcit

```
' this program prints the square of a number
Dim number As Short, num1 As Short, num2 As Short, result As Short
Dim letterSTR As String
Dim okTF As Boolean
SquareIt()
Print ""
Print "Finished"
```

```
End
'
Sub SquareIt()
WhatNumber()
Calcit()
End Sub
'
Sub WhatNumber()
input "Please, enter a number ", letterSTR
number = cint(letterSTR)
End Sub
'
Sub Calcit()
If number = 0 Then
Print "The square of zero is zero, of course"
Else
If number > 15 Then
TestDigits()
Else
Print "The number must be > 15"
okTF = False
End If
End If
If okTF Then
Print "The square of " + number + " is " + result
Else
Print "This is not as intended"
End If
End Sub
'
Sub TestDigits()
Dim cipherSTR, leftnumSTR As String
cipherSTR = Str$(number)
If Right$(cipherSTR, 1) <> "5" Then
Print "The number must end with 5"
okTF = False
Else
leftnumSTR = Left$(cipherSTR, (Len(cipherSTR) - 1))
num1 = Val(leftnumSTR)
num2 = num1 + 1
result = (num1 * num2 * 100) + 25
okTF = True
End If
End Sub
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Advice given by the author of Kbasic:

do not use ( ) to access arrays, better use [ ]

do not use 'Option OldBasic' or 'Option VeryOldBasic'

do not use 'On Error Goto', better use 'Try Catch'

do not use 'Nothing', better use 'Null'

avoid the use of the data type 'Variant'

do not use 'class_initialisize', better use 'Constructor', the same for the destructor

do not use 'Call' when calling a sub or function

do not use 'Optional' and 'IsMissing' with arguments in subs or functions, better use the default value of an argument

always use 'Do While…Loop' and 'Do …Loop While' instead of the other loops

always write many comments in your source code

use in conditions 'AndAlso' and 'OrElse' instead of the binary operators 'And' and 'Or'

avoid the use of 'ByRef' with primitive data types to get faster execution speed

do not use 'Data' and 'Def*', like 'DefInt'

use enumerations instead of many integer constants

use constants instead of the use of numeric literals many times in your source code
avoid the use of ‘GoSub’, better use real functions or real subs
avoid the use of ‘GoTo’, better use loops and other control flow language elements
‘Let’ and ‘Set’ are not needed
use ‘For Each’, when it is possible
use ‘For i As Integer = 0 To 10 Next’, instead of declaring the counter variable outside of the loop
name the variable names without giving them suffixes
always use ( ) when you call a sub or function

Footnotes

1. **www.kbasic.com**
2. I am well aware that there are other ways to draw a square. Here the important point is learning modular, structured programmming, not how to draw a square.
3. The adres programs are also included for educational purposes.
4. The different programs about character functions are adapted from a Forum contribution by John Antony Oliver (Microsoft Community Contributor)