+++ ADOK SOFT PRESENTS +++ ADOK SOFT PRESENTS +++ ADOK SOFT PRESENTS +++

THE REAL ADOK'S WAY TO

Creative Basic

A NEW METHOD STEP BY STEP

WRITTEN BY CLAUS-DIETER VOLKO, VIENNA

Homepage: http://www.hugi.scene.org/adok/
E-Mail: cdvolko@gmx.net

TABLE OF CONTENTS

All example programs are included as *CBA* files! They can be loaded into Creative Basic and immediately started.

CHAPTER 1

First steps

There are apparently many people who want to learn programming in Basic and are waiting for a tutorial. That's why I decided to start one.

My tutorial is about the interpreter Creative Basic, which can be downloaded for free at the Internet address http://www.ionicwind.com/

Let's start with the tutorial.

After installing Creative Basic you can start it in the Program menu. After starting it the editor appears. By clicking "File", "New" and "Source File", you can write a program.

Type the following program just like you'd type a letter. The name of the program is *TUTO0001*.

```
DIM name AS STRING
PRINT "Hello, majesty! What should I call you?"
PRINT "Enter your name!"
INPUT name
PRINT "Hello, " + name + ", I will fulfil all your wishes."
PRINT "I will show you now using what commands you can control me."
```

Once you've finished typing this program, you can run it by pressing F4.

First the following text appears: *Hello, majesty! What should I call you?*

Now it's your turn. Enter your name, a pseudonym or whatever. By pressing Enter you conclude the input.

Another text appears: *I will show you now using what commands you can control me.* Afterwards the message <<< *Program ended. Press any key to close* >>> appears. After another keypress the program ends.

That was our first program! It's as easy as that. Cool, isn't it? But how does the program work?

The first line of the program is *DIM name AS STRING*. In contrast to other, older Basic dialects variables must be defined in Creative Basic. Don't be confused, it's simple. Variables are containers in which values are stored. We define a variable called *name*. Its type is *STRING*. This makes *name* a variable in which texts can be stored.

The second line begins with *PRINT*.That is a command. The *PRINT* command has several uses as it can be called with parameters. *"Hello, majesty! What should I call you?"* is such a parameter.

PRINT is used to display strings or number on screen. In our example PRINT displays the following thing: *Hello, majesty! What should I call you?* This is a string. In Basic, strings must always be between quotation marks. Otherwise Basic would interpret this text as a variable.

We already understand the third line. Let's go to the fourth line. Here we find the *INPUT* command. What's that? With *INPUT*, strings can be read from the keyboard. In this case the computer asked you for your name. The sentence is displayed on the screen, and with *INPUT* the computer expects that you enter your name. As the parameter of *INPUT* this example program uses *name* – it's the variable we defined at the beginning.

The fifth line is yet another *PRINT* command. This time the parameter is several strings connected by the plus sign ("+"). One of these strings is the variable *name*.

The last line contain one more *PRINT* command.

We're done! That was the whole program. Now we can lean back and relax.

I'd like to give you some tips for this tutorial. It's a good thing to print the example programs and read the listing on paper. This can be done by clicking *File-Print*.

The program can be stored by clicking *File-Save*. When you're saving a program the first time, you'll be asked for the filename. For this program write *TUTO0001*. Creative Basic automatically adds the file extension *CBA*. The complete filename after saving will be *TUTO0001.CBA*.

After the end of a chapter you should experiment with the new comands. For this reason I sometimes give you "home exercises". The correct solutions are presented afterwards.

Exercise 1: Create a program that first asks the user for his name and then for his pseudonym. Afterwards the computer shall display the following text on screen: *"Good day, <name>! Your pseudo is <pseudo>. Welcome in the Creative Basic Programmers Club!"* <*name*> is for the variable in which the name will be stored, and <*pseudo*> is for the variable in which the pseudonym will be stored. In my case the sentences would be: *"Good day, Claus-Dieter Volko! Your pseudo is The Real Adok. Welcome in the Creative Basic Programmers Club!"* Mind that in contrast to the example *TUTO0001* you have to use two string variables.

Exercise 2: Now write a program which asks the user for his name and the name of his girlfriend. Afterwards it shall display a funny sentences about the two of them, like: *"<girlfriend>: 'When are you going to marry me, <user>?'"*

Solutions for Chapter 1

Exercise 1: Here you had to write a program that firsts asks the user for his name and then for his pseudo. Afterwards it displays a text. A possible solution is the following listing:

```
DIM name AS STRING
DIM pseudo AS STRING
PRINT "What's your name?"
INPUT name
PRINT "And what's your pseudo?"
INPUT pseudo
PRINT "Good day, " + name + "! Your pseudo is " + pseudo + "."
PRINT "Welcome in the Creative Basic Programmers Club!"
```

Exercise 2: You had to write a program that reads the names of the user and of his girlfriend. Afterwards a funny text was to be displayed.

Here is a possible solution:

```
DIM user AS STRING
DIM girlfriend AS STRING
PRINT "What's your name?"
INPUT user
PRINT "And what’s the name of your girlfriend or boyfriend?"
INPUT girlfriend
PRINT girlfriend + ": 'When are you going to marry me, " + user + "?'"
```

CHAPTER 2

Basic commands for processing numbers

Last time we learned the *PRINT* and *INPUT* commands. *PRINT* and *INPUT* are the basic commands for string processing. Now let's stake a look at number variables like we know them from mathematics.

Here's an example program. Enter it in Creative Basic and save it as *TUTO0002.CBA*.

```
DIM number1 AS INT
DIM number2 AS INT
PRINT "Enter number 1!"
INPUT number1
PRINT "Enter number 2!"
INPUT number2
PRINT "Addition:"
PRINT STR$(number1) + " +" + STR$(number2) + " =" + STR$(number1 + number2)
PRINT "Subtraction:"
PRINT STR$(number1) + " -" + STR$(number2) + " =" + STR$(number1 - number2)
PRINT "Multiplication:"
PRINT STR$(number1) + " *" + STR$(number2) + " =" + STR$(number1 * number2)
PRINT "Division:"
PRINT STR$(number1) + " /" + STR$(number2) + " =" + STR$(number1 / number2)
```

And now we have a simple calculation program! Start it and Creative Basic will show you basic calculation operations!

number1 and *number2* are, as some of you will probably already think, number variables. We defined them as *INT*, that is integer numbers – numbers greater than zero or equal zero, without digits behind the comma.

In order to display the calculations with *PRINT*, we must convert the numbers to strings. This is what the function *STR$* is for.

The symbols '+', '-', '*' and '/' are, as you've certainly already noticed, calculation symbols. '/' is for division.

Of course you cannot just display the results of the calculations but you can also pass them to variables. Here's another example program. Save it as *TUTO0003.CBA* and run it with *F4*.

```
DIM number1 AS INT
number1 = 100
DO
 number1 = number1 + 1
 PRINT number1
UNTIL number1 = 200
```

Now I want to describe this program line by line. Line 1 defines the variable *number1*, line 2 sets this variable to 100.

I forgot to tell you about this: The equal character ('=') can mean several different things in Basic. Here it means that the variable on the left is set to the value on the right.

If you write

```
100 = number1
```

you'll get an error message.

By the way, it's also possible to write *LET* before the variable. Our example would look like this:

```
LET number1 = 100
```

But using *LET* is neither necessary nor recommended nowadays.

The next line contains a new keyword. It's *DO*, which is a command which we'll learn more about in the chapter about loops and conditions. Now let's only say that the program code between *DO* and *UNTIL* is repeatedly executed until the condition next to *UNTIL* (*number1 = 200*) is fulfilled.

In the fourth line the variable *number1* is set to the result of the addition *number1 + 1*. This is easier than it sounds since it only means that *number1* is increased by one. Afterwards the number is displayed on screen, and with *UNTIL* everything gets repeated starting with the fouth line. Is everything clear?

CHAPTER 3

Simple graphical effects

How can you implement simple graphical effects to make your programs look better? There are a couple of commands for that which I’ll now list and explain.

It’s perhaps not so beautiful if the texts come one after another in a program. To change this there’s the *LOCATE* command. It allows you to place the cursor at any location inside the screen you wish. The syntax of this command is:

```
LOCATE row, column
```

The text mode in which we’re currently working has a resolution of 80 columns by 25 rows. With the *LOCATE* command we relocate the cursor to the position given by *row* and *column*. The next output will occur at this position.

To demonstrate the abilities of the *LOCATE* command, here’s an extended version of our calculation program:

```
DIM number1 AS INT
DIM number2 AS INT
LOCATE 2, 2
PRINT "What’s the first number?"
LOCATE 3,2
INPUT number1
LOCATE 5, 2
PRINT "What’s the second number?"
LOCATE 6,2
INPUT number2
CLS
LOCATE 2, 2
PRINT "Number 1:" + STR$(number1)
LOCATE 2, 60
PRINT "Number 2:" + STR$(number2)
LOCATE 3, 1
PRINT " ----------------------------------------------------------------------"
LOCATE 5, 2
PRINT "Addition:"
LOCATE 5, 18
PRINT number1 + number2
LOCATE 6, 2
PRINT "Subtraction:"
LOCATE 6, 18
PRINT number1 - number2
LOCATE 7, 2
PRINT "Multiplication:"
LOCATE 7, 18
PRINT number1 * number2
LOCATE 8, 2
PRINT "Division:"
LOCATE 8, 18
PRINT number1 / number2
```

A new command appearing in this program is *CLS*. *CLS* “clears“ the screen.

We’ll modify this program in the future. So save it as *TUTO0004.CBA*.

Until now all programs have used the same colours: white text on a black background. As this can get boring with time, Creative Basic offers a command to change the colours: *COLOR*. Its syntax is:

```
COLOR foregroundcolour, backgroundcolour
```

foregroundcolour is the colour in which the text will be printed, while *backgroundcolour* is, as the name says, the colour of the background.

Unfortunately you can't simply write *COLOR "yellow", "blue"* or something like that. You have to use colour codes. A colour code is a number between 0 and 31.

Every number between 0 and 7 is a "dark" colour. By adding 8, you get the corresponding "bright" colour. For example, the colour code of dark blue is 1. Therefore the colour code of bright blue is 9. The colour with the number 8 is the only exception – it's gray.

By adding 16 to a number, you get a "blinking" colour.

There's no way but learn the colour codes by heart. But as I've explained the system, it will be not so hard for you. Here's a complete table of the colour codes:

0 black	8 gray	16 blink black	24 blink gray
1 dark blue	9 bright blue	17 blink d. blue	25 blink b. blue
2 dark green	10 bright green	18 blink d. green	26 blink b. green
3 dark cyan	11 bright cyan	19 blink d. cyan	27 blink b. cyan
4 dark red	12 bright red	20 blink d. red	28 blink b. rot
5 dark purple	13 bright purple	21 blink d. purple	29 blink b. purple
6 brown	14 yellow	22 blink brown	30 blink yellow
7 white	15 bright white	23 blink white	31 blink b. white

Now let's take a look at our example program. It's a further development of *TUTO0004.CBA*, and that's why we store it as *TUTO0005.CBA*.

```
DIM number1 AS INT
DIM number2 AS INT
DIM blue AS INT
DIM yellow AS INT
DIM white AS INT
DIM bright AS INT
blue = 1
yellow = 14
white = 7
bright = 8
COLOR bright + white, blue
CLS
LOCATE 2, 2
PRINT "What's the first number?"
COLOR yellow, blue
LOCATE 3,2
INPUT number1
COLOR bright + white, blue
LOCATE 5, 2
PRINT "What's the second number?"
COLOR yellow, blue
LOCATE 6,2
INPUT number2
CLS
COLOR bright + white, blue
LOCATE 2, 2
PRINT "Number 1:"
COLOR yellow, blue
LOCATE 2, 9
PRINT STR$(number1)
COLOR bright + white, blue
LOCATE 2, 60
PRINT "Number 2:"
```

```
COLOR yellow, blue
LOCATE 2, 67
PRINT STR$(number2)
COLOR bright + white, blue
LOCATE 3, 1
PRINT " ----------------------------------------------------------------------"
LOCATE 5, 2
PRINT "Addition:"
COLOR yellow, blue
LOCATE 5, 18
PRINT number1 + number2
COLOR bright + white, blue
LOCATE 6, 2
PRINT "Subtraction:"
COLOR yellow, blue
LOCATE 6, 18
PRINT number1 - number2
COLOR bright + white, blue
LOCATE 7, 2
PRINT "Multiplication:"
COLOR yellow, blue
LOCATE 7, 18
PRINT number1 * number2
COLOR bright + white, blue
LOCATE 8, 2
PRINT "Division:"
COLOR yellow, blue
LOCATE 8, 18
PRINT number1 / number2
```

As you see, *CLS* doesn’t always make the screen black, but rather fills it using the current background colour.

The program is already quite complicated, isn’t it? In order to make it more easy to understand, you can insert comments. That is done by using the command *REM*. Everything in the line after this word is ignored. This can be used to explain program sections or to test different variants (simply write *REM* at the beginning of the program line – the consequence: commands in this line will be ignored).

The *REM* command can be abbreviated by an apostrophe. In order to write a comment at the end of a program line, Creative Basic (in contras to other Basic dialects) requires you to write a colon before it. By the way, the colon can be generally used to write several commands in one program line.

A little example (*TUTO0006.CBA*):

```
'Variable definitions
DIM blue AS INT
DIM yellow AS INT

'Variable assignments
blue = 1 : 'Colour blue
yellow = 14 : 'Colour yellow

'Main program
COLOR yellow, blue : 'Select colours
CLS : 'Clear screens
PRINT "Hi guys!" : 'Output text
REM PRINT "What a crappy weather there is today!" : 'Excluded command
PRINT "It’s beautiful weather today!!" : 'Output text
PRINT "See you tomorrow!" : 'Output text
```

Everything clear?

CHAPTER 4

Blocks

Are you ready for a different kind of experience? It's our first block statement. This means it consists of two commands.

We're talking about *DO/UNTIL*. We once got to know it in an example program.

Everything between *DO* and *UNTIL* will first be executed once. Then the program checks whether the condition next to *UNTIL* is fulfilled. If it isn't, the code between *DO* and *UNTIL* will be executed again – and so on.

An example:

```
DIM i AS INT
i = 0
DO
 PRINT "I'm such a handsome guy"
 i = i + 1
UNTIL i = 10
```

This program displays the text *I'm such a handsome guy* exactly ten times.

For the sake of clarity, I've indented the lines between *DO* and *UNTIL*. You should get used to indenting the code between the beginning and the end of a block as otherwise it's getting confusing.

Blocks can be nested. An example:

```
DIM i AS INT
DIM time AS INT
i = 0
DO
 PRINT "I'm such a handsome guy"
 i = i + 1
 time = TIMER
 DO
 UNTIL TIMER > time + 0.5
UNTIL i = 10
```

TIMER returns the number of seconds since midnight. Our code results in a break of half a second between two instances of displaying the text *I'm such a handsome guy*.

The innermost *UNTIL* belongs to the innermost *DO*, the second innermost *UNTIL* to the second innermost *DO* and so on till the outermost *UNTIL*, which belongs to the outermost *DO*.

In theory, blocks can be nested an infinite number of times! Here, however, we only used two "levels".

In Creative Basic there's another block which is quite similar: it's the *WHILE* block. With *WHILE*, our program looks like this:

```
DIM i AS INT
DIM time AS INT
i = 0
WHILE i < 10
 PRINT "I'm such a handsome guy"
 i = i + 1
 time = TIMER
 DO
 UNTIL TIMER > time + 0.5
```

```
ENDWHILE
```

In contrast to *DO* the condition for *WHILE* is at the beginning of the block. *WHILE* means "as long as". As long as the variable *i* is lower than 10, the code between *WHILE* and *ENDWHILE* will be repeatedly executed. Attention: If you didn't set *i* to 0 before the block, but to 10 or to an even greater value, the code in the block wouldn't be executed at all since the condition would not be met right at the beginning. (Try it!) That's different from the *DO* block: In the *DO* block the code in the block is always executed at least once because the condition is only checked at the end of the block.

The *DO* and *WHILE* blocks are also called loops, by the way.

In Basic there's yet another loop which is perfect for counting purposes: it's the *FOR/NEXT* loop. Here's an example program:

```
DIM i AS INT
DIM time AS INT
i = 0
FOR i = 1 TO 9 STEP 1
 PRINT "I'm such a handsome guy"
 time = TIMER
 DO
 UNTIL TIMER > time + 0.5
NEXT i
PRINT "i now has the value:" + STR$(i)
```

Well, isn't it more compact!

The fourth line contains the beginning of the block (like *DO*) and the ninth the end of the block (like *UNTIL*).

In the fourth line we write that the variable *i* shall be used as the counter. The start value shall be 1 and the end value 9. *STEP* is the step width. If you skip *STEP*, it will be assumed as 1.

What is the beginning of the loop doing? The first time it's executed, the loop counter will be intialized with the start value. Afterwards all the commands between *FOR* and *NEXT* will be executed. *NEXT* increases the counter by the step width. If the counter equals the ending value or is even greater than it, the loop will be stopped and the next command after the loop will be executed. Otherwise it will jump back to *FOR* and the commands between *FOR* and *NEXT* will be executed once again.

You see, it's all similar to *DO/UNTIL*. Even nesting is possible. It's even possible to nest different types of loops!

That also goes for the *IF/THEN/ELSE/ENDIF* block although this isn't a loop but a conditional statement. We'll talk about this statement now.

The *IF/THEN/ELSE/ENDIF* block has the purpose to create branches. If a certain condition is fulfilled, the part until *ENDIF* will be executed. Otherwise the program will jump to the command after *ENDIF*.

The condition is usually a comparison of two variables or a variable and a value. An example program demonstrates the possible variants.

```
'Comparisons
DIM Number AS INT
COLOR 14, 1
CLS
PRINT "Enter a number!"
INPUT Number
PRINT "This number fulfills the following conditions:"
IF Number < 100
 PRINT "It is lower than 100."
```

```
ENDIF
IF Number <= 100
 PRINT "It is lower than or equal to 100."
ENDIF
IF Number = 100
 PRINT "It is equal to 100."
ENDIF
IF Number >= 100
 PRINT "It is greater than or equal to 100."
ENDIF
IF Number > 100
 PRINT "It is greater than 100."
ENDIF
```

Now the comparison operators should be clear.

The program could also be written the following way:

```
'Comparisons
DIM Number AS INT
COLOR 14, 1
CLS
PRINT "Enter a number!"
INPUT Number
PRINT "This number fulfills the following conditions:"
IF Number < 100 THEN PRINT "It is lower than 100."
IF Number <= 100 THEN PRINT "It is lower than or equal to 100."
IF Number = 100 THEN PRINT "It is equal to 100."
IF Number >= 100 THEN PRINT "It is greater than or equal to 100."
IF Number > 100 THEN PRINT "It is greater than 100."
```

This variant of *IF* is a bit more compact as it doesn't require *ENDIF*. But you have to write everything in a line.

There are also the nice little operators *&* and **|**. Using them you can connect several conditions with each other.

If two conditions are connected by *&*, the code between the condition and *ENDIF* is only processed if both conditions are fulfilled. Right, that can also be achieved by two nested *IF/THEN/ELSE/ENDIF* blocks!

With **|** the code is already processed if only one of the requirements is fulfilled.

In order to demostrate it, here's another example program:

```
'Comparisons with & and |
DIM Number1 AS INT
DIM Number2 AS INT
COLOR 14, 1
CLS
PRINT "Enter the first number!"
INPUT Number1
PRINT "Enter the second number!"
INPUT Number2
PRINT "The following conditions are fulfilled:"
IF Number1 = 100 | Number2 = 100
 PRINT "At least one of the two numbers is 100."
ENDIF
IF Number1 = 100 & Number2 = 100
 PRINT "Both numbers are 100."
ENDIF
IF Number1 <> 100 & Number2 <> 100
 PRINT "Neither of the two numbers is 100."
```

```
ENDIF
```

Ah yeah, 'not equal to' is written as '<>' in Creative Basic. I forgot to tell you.

What about the *ELSE* clause? If the condition isn't fulfilled, the execution is usually continued with the next command after *ENDIF*. But if an *ELSE* clause exists, the code between *ELSE* and *ENDIF* will be executed before that!

It's difficult to explain it using words, but in reality it's not difficult at all. Let's try an example program (it's already *TUTO0014.CBA*!).

```
'Comparisons with ELSE
DIM Number1 AS INT
DIM Number2 AS INT
COLOR 14, 1
CLS
PRINT "Enter a number!"
INPUT Number1
PRINT "Enter another number!"
INPUT Number2
PRINT
IF Number1 = Number2
 PRINT "The two numbers are the same."
ELSE
 PRINT "The two numbers are NOT the same."
ENDIF
```

For fun, I've optimized this program. The optimized version isn't necessary to understand this tutorial, but it might be interesting.

```
'Comparisons with ELSE
DIM Number1 AS INT
DIM Number2 AS INT
Dim Ausgabe AS STRING
COLOR 14, 1
CLS
PRINT "Enter a number!"
INPUT Number1
PRINT "Enter another number!"
INPUT Number2
PRINT
Ausgabe = "The two numbers are "
IF Number1 <> Number2 THEN Ausgabe = Ausgabe + "NOT "
PRINT Ausgabe + "the same."
```

That was the *IF/THEN/ELSE/ENDIF* block.

As said, it can be nested like any block. This nesting is extremely important for the structurized programming of adventure games. At the end of this chapter an example will follow, but let's first talk about the *SELECT/CASE b*lock.

If you want to quickly check a variable for several values, this block is far better than an *IF/THEN/ELSE/ENDIF b*lock. Instead of writing *'IF ... '*, you write *SELECT Variable*. *Variable* stands for the name of the variable the comparisons are about.

A comparison is always started with *CASE*. Next to it comes the value to which the variable mentioned at the beginning of the block shall be compared.

If the variable is equal to one of the values next to *CASE*, the code after *CASE* is executed, until the next *CASE*. Otherwise the program jumps to the next *CASE*. The *SELECT/CASE* block ends with *ENDSELECT*.

Let's check out an example that demonstrates how it works. It shows well how to implement the processing of the input in adventure games with the *SELECT/CASE* block.

```
'Simple adventure game with SELECT/CASE
DIM InputString AS STRING
COLOR 14, 1
CLS
PRINT "Welcome to the Mysterious Creative Basic Adventure. You must always enter"
PRINT "the number that is next to the selected option. 'oK' stands for any other key."
PRINT
PRINT "You're standing in front of a castle. Do you dare enter? "
PRINT
PRINT " ( 1) Yes"
PRINT " (oK) No"
PRINT
INPUT InputString
CLS
SELECT InputString
CASE "1"
 PRINT "Now you're inside the castle. You can choose whether to go westwards or"
 PRINT "eastwards. Of course you can also leave the castle (coward!)."
 PRINT "Your decision, noble knight?"
 PRINT
 PRINT " ( 1) West"
 PRINT " ( 2) East"
 PRINT " (oK) Leave castle"
 PRINT
 INPUT InputString
 CLS
 SELECT InputString
 CASE "1"
  PRINT "There's an evil dragon which you, noble knight, make mincemeat of with"
  PRINT "your sword! You've saved the princess and as a reward you may marry her"
  PRINT "(and have a lot of fun with her)! End."
  END
 CASE "2"
  PRINT "Oh, you poor hero. Once gone to the east, you fall down a dark abyss!"
  PRINT "End."
  END
 DEFAULT
  PRINT "After leaving the castle, the door is closing behind you."
  PRINT "You've lost your chance to win, you'll never get in again! End."
  END
 ENDSELECT
DEFAULT
 PRINT "Oh well, then not. Better run Excel and do boring table calculations! End."
ENDSELECT
```

Nesting is demonstrated here. By the way, *DEFAULT* corresponds to *ELSE* from the *IF/THEN/ELSE/ENDIF* block, and *END* ends program execution. Now the same with the *IF/THEN/ELSE/ENDIF* block.

```
'Simple adventure game with IF/THEN/ELSE/ENDIF
DIM InputString AS STRING
COLOR 14, 1
CLS
PRINT "Welcome to the Mysterious Creative Basic Adventure. You must always enter"
PRINT "the number that is next to the selected option. 'oK' stands for any other key."
PRINT
PRINT "You're standing in front of a castle. Do you dare enter? "
PRINT
PRINT " ( 1) Yes"
PRINT " (oK) No"
PRINT
INPUT InputString
CLS
IF InputString = "1"
 PRINT "Now you’re inside the castle. You can choose whether to go westwards or"
 PRINT "eastwards. Of course you can also leave the castle (coward!)."
 PRINT "Your decision, noble knight?"
 PRINT
 PRINT " ( 1) West"
 PRINT " ( 2) East"
 PRINT " (oK) Leave castle"
 PRINT
 INPUT InputString
 CLS
 IF InputString = "1"
  PRINT "There’s an evil dragon which you, noble knight, make mincemeat of with"
  PRINT "your sword! You’ve saved the princess and as a reward you may marry her"
  PRINT "(and have a lot of fun with her)! End."
  END
 ENDIF
 IF InputString = "2"
  PRINT "Oh, you poor hero. Once gone to the east, you fall down a dark abyss!"
  PRINT "End."
  END
 ELSE
  PRINT "After leaving the castle, the door is closing behind you."
  PRINT "You’ve lost your chance to win, you’ll never get in again! End."
  END
 ENDIF
ELSE
 PRINT "Oh well, then not. Better run Excel and do boring table calculations! End."
ENDIF
```

As you see, the listing a bit shorter (one line), but more difficult to read! But in the end it’s up to the programmer whether he uses *SELECT/CASE* or *IF/THEN/ELSE/ENDIF*. I personally prefer *SELECT/CASE*.

CHAPTER 5

Programming adventure games

If you've studied well so far, you fulfill all the requirements to get creative! We will learn a couple of new techniques which can also be useful in other programming tasks.

If you want to create an adventure game, you should first get a rough overview about the story. Usually an adventure is a continuous story which allows intervention by the player at some stages. In professional adventures there are so many opportunities to intervene that everything becomes a game.

Programs with less intervention opportunities are called interactive films. Most of them are so primitive that the player can only press a cursor key now and then and hope that it was the right one!

Our first game is supposed to have more intervention opportunities than interactive films, but less than professional adventures.

Often objects are used in adventures. We can store them in variables. If the value of the variable is 1, the person has the object, if it is 0, then she doesn't. We could also other other values, e.g. 2: the object has already been used, but is still owned by the player. There are no limits to the programmer's imagination (except maybe the memory)!

As shown in the example from the previous chapter, the options are displayed with *PRINT*. Afterwards the decision is read from the keyboard and it's evaluated with *SELECT/CASE*.

In order to make the input more comfortable, we assign a number to each option. The user then only has to enter the number that is assigned to the respective option. For the last option any number or string can be written.

Moreover, we often have to "jump" in the program. For instance, if we enter a house and then leave it, the program has to jump to the code location that reflects the situation that we're standing in front of the house.

The easiest way to implement such jumps is using the *GOTO* command. To do so, we have to define a so-called label, to which *GOTO* can then jump. A label consists of a string, which this time isn't written in between quotation marks, and a colon. It must be written in a line of its own.

After *GOTO* you must write the name of the labels as the parameter as it's of course possible to use more than one label in a program. If there are several labels, all the labels must have different names! (I know from experience that many things which are self-evident for me nowadays are quite hard to understand for beginners. That's why I rather explain things more detailed than necessary than get mailed a load of questions.)

Now that you know the theory, I want to provide you with a little example.

```
'Journey to Moon
'Strongly abbreviated version for The Real Adok's Way to Creative Basic
DIM a AS STRING
DIM Inpu AS STRING
DIM i1 AS STRING
DIM TalkedToNele AS INT
DIM Joystick AS INT
COLOR 14, 1
CLS
'Intro
PRINT "Journey to Moon - The Gerhard-driven space shuttle"
PRINT "Copyright (C) 1995-1996 by Adok Soft"
PRINT
PRINT "Strongly abbreviated version for The Real Adok's Way to Creative Basic"
```

```
INPUT a
CLS
PRINT "Clausi, Stefan, Philipp, Franzi, Gerhard, Marius, Zahra and Nele want to"
PRINT "fly to the moon! Clausi, the genius, constructed a space shuttle called"
PRINT "Astrein Shuttle and declared himself commander. The special thing about the"
PRINT "Astrein Shuttle is certainly not ist speed, which is only a couple of"
PRINT "megameters per second, but the drive. It works this way: Gerhard attaches"
PRINT "himself to the shuttle and has Zahra feed him. The more he eats, the more"
PRINT "fumes he produces. And these fumes are what drive the shuttle!"
INPUT a
CLS
PRINT "Once again: The staff is "
PRINT
PRINT "Clausi.......... Commander"
PRINT "Franzi.......... 1st Officer"
PRINT "Philipp......... 2nd Officer"
PRINT "Stefan.......... 3rd Officer"
PRINT "Marius.......... Head biologist and toilet supervisor"
PRINT "Nele............ Navigator"
PRINT "Gerhard......... Fuel producer"
PRINT "Zahra........... Nurse"
INPUT a
CLS
PRINT "You assume the role of Clausi, the commander. You and your staff have"
PRINT "already been travelling in space for a very long time (one minute!)."
PRINT "Your task is to safely guide your staff and yourself to the moon."
PRINT "Easy? Sounds so. But it isn't..."
INPUT a
'The game starts!

MainhallStart:
CLS
PRINT "You're now in the main hall of the Astrein shuttle. From here you can"
PRINT "enter all rooms of the shuttle and you can even leave the shuttle."
PRINT "Before doing the latter thing, I'd recommend getting the appropriate"
PRINT "equipment..."
GOSUB InputString
CLS
SELECT Inpu
CASE "1"
 PRINT "There is nobody you could talk to!"
CASE "2"

MhRetry1:
 PRINT "What do you want to look at?"
 PRINT " (1) The doors"
 PRINT " (2) The room"
 PRINT " (3) The air"
 PRINT " (4) The plate"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  PRINT "There are five big doors which are shut by a laser field. If you touch"
  PRINT "the laser field, it gets automatically deactivated."
 CASE "2"
  PRINT "A large, empty room which is called the main hall. At the walls old"
  PRINT "butter breads are sticking, and the five doors are shut by a laser field."
  PRINT "If you compare this room with a certain class room of the Goethe high school,"
  PRINT "a certain similarity can be noticed."
 CASE "3"
  PRINT "Das seeing organ claims, it were a transparent nothing. The smelling and"
  PRINT "the breathing organs know it better."
 CASE "4"
  PRINT "On it 'EXIT' is written. Obviously it leads to the exit of this shuttle."
  PRINT "But you, being the constructor of the space shuttle, should know that best!"
 DEFAULT
  GOTO MhRetry1
 ENDSELECT
CASE "3"

MhRetry2:
 PRINT "What do you want to take?"
 PRINT " (1) Laser field"
 PRINT " (2) Butter bread"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  PRINT "Are you crazy!? If you go on like that, you’ll never succeed!"
CASE "2"
  PRINT "Ouch! This cruel bread sticking to the wall makes you cringe!"
  PRINT "Better leave it there!"
 DEFAULT
  GOTO MhRetry2
 ENDSELECT
CASE "4"

MhRetry3:
 PRINT "Where do you want to go?"
 PRINT " (1) Door 1"
 PRINT " (2) Door 2"
 PRINT " (3) Door 3"
 PRINT " (4) Door 4"
 PRINT " (5) Door 5"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  PRINT "You can enter this room only in the unabbreviated version!"
 CASE "2"
  PRINT "You can enter this room only in the unabbreviated version!"
 CASE "3"
  GOTO CommandroomStart
 CASE "4"
  PRINT "You can enter this room only in the unabbreviated version!"
 CASE "5"
  GOTO OfficersroomStart
 DEFAULT
  GOTO MhRetry3
 ENDSELECT
DEFAULT
 GOTO MainhallStart
ENDSELECT
INPUT a
GOTO MainhallStart

CommandroomStart:
CLS
PRINT "This is the command room of the Astrein Shuttle. In front of you there is"
PRINT "a huge control panel, next to which Nele is sitting, who desparately tries"
PRINT "to handle it. Behind her there’s a giant window."
GOSUB InputString
CLS
SELECT Inpu
CASE "1"

CrRetry1:
 PRINT "Who do you want to talk to?"
 PRINT " (1) Nele"
 PRINT " (2) Autopilot"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  SELECT TalkedToNele
  CASE 1
   PRINT "Nele: 'Have you succeeded in activating the autopilot?'"
   PRINT "Clausi: 'I’m just about it!'"
  CASE 0
   PRINT "Clausi: 'Are there any problems?'"
   PRINT "Nele: 'Yes, massive ones! We are out of the path, and I cannot activate"
   PRINT " the autopilot any more!'"
   PRINT "Clausi: 'Then take a look in the manual!'"
   PRINT "Nele: 'I've lost it.'"
   PRINT "Clausi: 'Wonderful! In the end I must do everything myself. Let me try!'"
   TalkedToNele = 1
  ENDSELECT
 CASE "2"
  PRINT "Clausi: 'Dear Mister Autopilot, please fly us to the moon!'"
  PRINT
  PRINT "..........................."
  PRINT
  PRINT "No answer. Well, it isn't that easy!"
 DEFAULT
  GOTO CrRetry1
 ENDSELECT
CASE "2"

CrRetry2:
 PRINT "Whatdo you want to look at?"
 PRINT " (1) Nele"
 PRINT " (2) Control panel"
 PRINT " (3) Window"
 PRINT " (4) Carpet"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  PRINT "Nele, the navigator of the Astrein Shuttle, is sitting casually on her chair,"
  PRINT "listening to a new CD with her Discman and trying to find out how to activate"
  PRINT "the autopilot of the Astrein Shuttle."
 CASE "2"
  IF TalkedToNele = 1 & Joystick = 5
   PRINT "You enter the code Franzi told you."
   PRINT "Suddenly Nele shouts: 'Yeah! You've done it! You've activated the autopilot!"
   INPUT a
   CLS
   PRINT "That was the abbreviated version of 'Journey to Moon' for"
   PRINT "The Real Adok's Way to Creative Basic. See you in the Full Version!"
   END
  ELSE
   PRINT "The control panel of the Astrein Shuttle consists of a load of buttons,"
   PRINT "levers, switches, joysticks and joypads. Even a computer with many games"
   PRINT "is in it! What else do you think are the joysticks and joypads needed for"
   PRINT "than the entertainment of the staff members?"
  ENDIF
 CASE "3"
  PRINT "A huge window which permits a look at what's located outside of the shuttle!"
  PRINT "Hey, wait a moment! There are also Zahra and Gerhard!"
 CASE "4"
  PRINT "LOOK AT THE CARPET? How's that possible, there being no carpet?"
 DEFAULT
  GOTO CrRetry2
 ENDSELECT
```

```
CASE "3"

CrRetry3:
 PRINT "What do you want to take?"
 PRINT " (1) Carpet"
 PRINT " (2) Control panel"
 PRINT " (3) Joystick"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  PRINT "TAKE THE CARPET? How's that possible, there being no carpet?"
 CASE "2"
  PRINT "A little bit too heavy, isn't it? Yes, a little bit much too heavy!"
 CASE "3"
  IF Joystick < 2
   PRINT "You take one of the many joysticks from the control panel and put them in"
   PRINT "your bag. Maybe it will be of use some day?"
   Joystick = 2
  ELSE
   PRINT "You've already got one!"
  ENDIF
 DEFAULT
  GOTO CrRetry3
 ENDSELECT
CASE "4"

CrRetry4:
 PRINT "Where do you want to go?"
 PRINT " (1) Door"
 INPUT i1
 CLS
 IF i1 <> "1"
  GOTO CrRetry4
 ENDIF
 GOTO MainhallStart
DEFAULT
 GOTO CommandroomStart
ENDSELECT
INPUT a
GOTO CommandroomStart

OfficersroomStart:
CLS
PRINT "You are in the officers room. There are the three officers Franzi, Philipp"
PRINT "and Stefan. They are playing chess with the computer. (Do they know how to"
PRINT "play chess at all!?)"
GOSUB InputString
CLS
SELECT Inpu
CASE "1"

 OfrRetry1:
 PRINT "Who do you want to talk to?"
 PRINT " (1) Stefan"
 PRINT " (2) Franzi"
 PRINT " (3) Philipp"
 INPUT i1
 CLS
 SELECT i1
 CASE "1"
  PRINT "Stefan: 'Hmmm, should I sacrifice my king to save the queen?'"
 CASE "2"
  IF Joystick = 0
   PRINT "Franzi: 'Stefan, Philipp, don't be nasty! Let me participate in the game!'"
```

```
    PRINT "Philipp: 'We would need a third joystick for that, but we don't have it!'"
   ENDIF
   IF Joystick = 5
    PRINT "Franzi: 'Please give me the bag!'"
    PRINT "Clausi: 'I won't give you the bag!'"
    PRINT "Franzi: 'As I said: The code is up, right, down, left, up or the other"
    PRINT " way round!"
   ENDIF
   IF Joystick > 2 & Joystick < 5 & TalkedToNele <> 1
    PRINT "Franzi: 'Thanks a lot for the joystick, Clausi!'"
    PRINT "Clausi: 'It's my pleasure!'"
   ENDIF
   IF Joystick = 4 & TalkedToNele = 1
    PRINT "Clausi: 'Hey, Fratzi, how about this original Philippean plastic bag?"
    PRINT " It has fallen down from Philipp's bed!'"
    PRINT "Franzi: 'Really? An object is always good for a rumour! Give it to me!'"
    PRINT "Clausi: 'First tell me the code!'"
    PRINT "Franzi: 'Okay. Up, right, down, left, up ... or the other way round?'"
    PRINT "Clausi: 'You're a truly helpful boy! I won't give you the bag!'"
    Joystick = 5
   ENDIF
   IF Joystick > 2 & Joystick < 5 & TalkedToNele = 1
    PRINT "Clausi: 'Hey, Fratzi, do you know how to activate the autopilot?'"
    PRINT "Franzi: 'No! But I got the manual!'"
    PRINT "Clausi: 'Give it to me!'"
    PRINT "Franzi: 'Hm... Actually I ought to give it to you, as you're my commander"
    PRINT " and you also gave me the joystick. Nevertheless: I'll give you the manual"
    PRINT " only if you give me another present! Make me commander!"
    PRINT "Clausi: 'I won't do that!'"
    PRINT "Franzi: 'Then do something else!'"
    Joystick = 4
   ENDIF
   IF Joystick = 2
    PRINT "Franzi: 'Come on, Philipp, let me play!'"
    PRINT "Philipp: 'How often shall I tell you: We need a third joystick for that!"
    PRINT "Clausi: 'Like the one I'm holding in my hands?'"
    PRINT "Philipp: 'Yes, exactly like this one! Give it to us!'"
    PRINT "Clausi: 'Here you are.'"
    PRINT "Franzi: 'Thanks, Clausi! You are my best friend!'"
    Joystick = 3
   ENDIF
  CASE "3"
   PRINT "Philipp: 'Chess is truly a funny game! If I was only able to play it...'"
  DEFAULT
   GOTO OfrRetry1
  ENDSELECT
 CASE "2"

OfrRetry2:
  PRINT "What do you want to look at?"
  PRINT " (1) Chess computer"
  PRINT " (2) Franzi"
  PRINT " (3) Philipp"
  PRINT " (4) Stefan"
  INPUT i1
  CLS
  SELECT i1
  CASE "1"
   PRINT "A huge computer using which you can only play chess, but for three players!"
 CASE "2"
   PRINT "This chap is called Franzi and he is the first officer. His nicknames are"
   PRINT "Fratzi, Intrigator and FBI. The last thing means 'Fast blonde islander'."
   PRINT "Well, Marius, who invented this nickname, meant something else with the 'I',"
   PRINT "but we don't wanna be nasty."
  CASE "3"
   PRINT "This chap is called Philipp and he is the second officer. He beats the drums"
   PRINT "like others beat their dear classmates."
  CASE "4"
   PRINT "This chap is called Stefan and he is the third officer. This genius is also"
   PRINT "called Stex Mel. One of his most famous quotations: 'Philosophy is nothing"
   PRINT "but a mix of poetry and fiction.'"
  DEFAULT
   GOTO OfrRetry2
  ENDSELECT
 CASE "3"

 OfrRetry3:
  PRINT "What do you want to take?"
  PRINT " (1) Chess computer"
  PRINT " (2) Philipp's joystick"
  PRINT " (3) Franzi"
  INPUT i1
  CLS
  SELECT i1
  CASE "1"
   PRINT "You already have one at home!"
  CASE "2"
   PRINT "He won't give it to you!"
  CASE "3"
   PRINT "Clausi: 'Fratzi, let me embrace you!'"
   PRINT "Franzi: 'You want to tease me!?'"
  DEFAULT
   GOTO OfrRetry3
  ENDSELECT
 CASE "4"
  PRINT "Where do you want to go?"
  PRINT " (1) Door"

 OfrRetry4:
  INPUT i1
  IF i1 <> "1"
   GOTO OfrRetry4
  ENDIF
  GOTO MainhallStart
 DEFAULT
  GOTO OfficersroomStart
 ENDSELECT
 INPUT a
 GOTO OfficersroomStart

 InputString:
 PRINT " (1) Talk to"
 PRINT " (2) Look at"
 PRINT " (3) Take"
 PRINT " (4) Go to"
 PRINT " (5) Quit"
 INPUT Inpu
 IF Inpu = "5"
  END
 ENDIF
 RETURN
```

"That is supposed to be a small example?" some people will be asking themselves. Yes, with its 395 lines which occupy roughly 13 KB this example program is already very large. But compared to a full adventure game with everything but graphics, mouse control and sound it's rather tiny!

A few explanations to the example program:

• *Joystick* and *TalkedToNele* are variables in which we store how far you have come in the game, what you have done and so on.

• The *GOSUB* command corresponds to *GOTO*, but if Basic gets to *RETURN*, the program jumps back to the line in which *GOSUB* is written.

CHAPTER 6

Graphics programming

If you want to use graphics commands, you must first create a window. This is done by the *WINDOW* command like in this example program:

```
DIM Window1 AS WINDOW
WINDOW  Window1,  0,  0,  640,  480,  @MINBOX  +  @MAXBOX  +  @SIZE,  0,  "My  Window",
Window1Procedure
WAITUNTIL Window1 = 0
END

SUB Window1Procedure
 IF @CLASS = @IDCLOSEWINDOW
  CLOSEWINDOW Window1
 ENDIF
RETURN
```

We need a window variable, a variable of the type *WINDOW* (yes, the type is called like the command). We define it in the first line.

Then we create the window using the *WINDOW* command. This command eats a lot of parameters. What do we have to feed it with? First with the window variable, then with the co-ordinates of the pixel on the top left where the window shall be displayed. We always start counting with 0, the pixel (0, 0) is the one on the very top left of the screen. Experiment a bit by inserting other values and watching the effects. Next come the width and the height of the window, here I've chosen 640 and 480 pixels. The next parameter is a collection of properties. In this case I wrote *@MINBOX* + *@MAXBOX* + *@SIZE*. This makes the window get a box on the top right which you can click to minimize it (to make it as small as possible) and one to maximize it (to make it as large as possible). Moreover, it can be resized by clicking the border and dragging it. These three properties are connected by +. It's also possible to use | instead. If you need none of these properties, you write 0. The next parameter is a handle to the window that is the "parent window" of our window. A "child window" can only move within its parent window. We have no parent window, so we write 0. Then comes the title of the window, which is displayed on the top left – we just take *My Window*. The last parameter is the procedure which is responsible for the control of the window.

What are procedures? Procedures are sub-programs, that is parts of the program enclosed between the command *SUB* and *RETURN*.

Usually you call procedures from the program. However, here the procedure is not called from the program, but it's automatically called by the *WINDOW* command. The sense of this procedure is to detect whether certain events have happened, and if so, to react on them. Our program only reacts if the user has done something that leads to a closing of the window. If *@CLASS = @IDCLOSEWINDOW*, the window shall be closed – *CLOSEWINDOW* Window1.

In the main program the WINDOW command is followed by the line *WAITUNTIL Window1 = 0*. The program waits until the window variable takes on 0. It takes on this value when the window has been closed. The program interrupts its execution and waits until the procedure *Window1Procedure* indirectly sets the variable *Window1* to 0.

The *END* command ends the program. It must be written before *SUB* as otherwise an error would result.

Let's do something with our window. Let's draw lines!

```
'LINE-Demo
DIM Window1 AS WINDOW
WINDOW  Window1,  0,  0,  640,  480,  @MINBOX  +  @MAXBOX  +  @SIZE,  0,  "My  Window",
Window1Procedure
LINE Window1, 100, 100, 320, 200, RGB(255,0,0)
LINE Window1, 80, 400, 420, 400, RGB(0,0,255)
WAITUNTIL Window1 = 0
END

SUB Window1Procedure
 IF @CLASS = @IDCLOSEWINDOW
  CLOSEWINDOW Window1
 ENDIF
RETURN
```

The *LINE* command is for drawing lines. The first parameter is the window variable – *LINE* has to know where to paint the line. First comes the x co-ordinate, then the y co-ordinate. Next come the co-ordinates of the end point, again it's first x, then y. Finally the colour of the pixel. We work with the RGB colour model. RGB stands for "red, green, blue". Each of these three colour components has a value between 0 and 255. *RGB(255,0,0)* is red, *RGB(0,0,255)* is blue. Try mixing colours.

You can also leave out the colour. Then the line is drawn in black.

If you don't want to draw entire lines, but only want to set single pixels, the command *PSET* is for you. Syntax: *PSET WindowVariable, x, y, Colour*.

However, if you not only want to draw lines but full rectangles, *RECT* is the command of your choice. An example program demonstrates it:

```
'RECT-Demo
DIM Window1 AS WINDOW
WINDOW  Window1,  0,  0,  640,  480,  @MINBOX  +  @MAXBOX  +  @SIZE,  0,  "My  Window",
Window1Procedure
RECT Window1, 10, 200, 300, 200, RGB(0,0,255), RGB(255,255,0)
WAITUNTIL Window1 = 0
END

SUB Window1Procedure
 IF @CLASS = @IDCLOSEWINDOW
  CLOSEWINDOW Window1
 ENDIF
RETURN
```

The parameters are the window variable, the x and y co-ordinates of the top left pixel, the width and the height and the border colour and the fill colour. You can also leave out the last two parameters. If you leave out only the last parameter, the rectangle won't get filled. If you also leave out the penultimate parameter, the border colour is black.

The next command is *CIRCLE*. With this command you can draw circles. The parameters are the center point, the radius and the colour. As always, here's an example.

```
'CIRCLE-Demo
DIM CPointX AS INT
DIM CPointY AS INT
DIM Radius AS INT
DIM Window1 AS WINDOW
CPointX = 320
CPointY = 200
Radius = 100
WINDOW  Window1,  0,  0,  640,  480,  @MINBOX  +  @MAXBOX  +  @SIZE,  0,  "My  Window",
Window1Procedure
CIRCLE Window1, CPointX, CPointY, Radius, RGB(0,0,255), RGB(255,255,0)
LINE Window1, CPointX - Radius, CPointY, CPointX + Radius, CPointY
LINE Window1, CPointX, CPointY - Radius, CPointX, CPointY + Radius
WAITUNTIL Window1 = 0
END
```

```
SUB Window1Procedure
 IF @CLASS = @IDCLOSEWINDOW
  CLOSEWINDOW Window1
 ENDIF
RETURN
```

Like with *RECT*, border and fill colours can be left out.

You can also print text in a window. You use the *PRINT* command for this. But you have to write in what window the text is to be printed. So you write e.g.:

```
PRINT Window1, "Hello World"
```

Instead of *LOCATE* you use the command *MOVE* in the window. The first parameter is the window, the second the x and the third the y coordinate. It's something like that:

```
MOVE Window1, 80, 100
```

CHAPTER 7

Data types in Creative Basic

Data types are somewhat like the number sets in mathematics.

Until now we have got to know two data types: integer numbers (*INT*) and strings (*STRING*).

There are also data types for floating point numbers. One of them is *FLOAT*. It has only a limited number of digits before and behind the comma. A higher number of digits is supported by the data type *DOUBLE*.

The *DIM* command also allows the definition of so-called arrays. If you write:

```
DIM ArrayName[x] AS INT
```

an array will be created that consists of *x INT* variables. Each of these variables can be addressed as if it wasn't part of an array. The first variable is *ArrayName[0]*, the second *ArrayName[1]* and so on. For instance, if you want to assign the value 10 to the fifth variable of the array *FullNumber*, you write:

```
FullNumber[4] = 10
```

To make it understandable, here's an example program:

```
'Array-Demo
DIM Summand[3] AS FLOAT
DIM Sum AS FLOAT
DIM Counter AS INT
COLOR 14, 1
CLS
FOR Counter = 0 TO 2
 PRINT "Summand" + STR$(Counter + 1)
 INPUT Summand[Counter]
 Sum = Sum + Summand[Counter]
NEXT Counter
PRINT
PRINT STR$(Summand[0]) + " +" + STR$(Summand[1]) + " +" + STR$(Summand[2]) + " =" +
STR$(Sum)
```

Since this example program is a bit more complicated than the others, we want to discuss it line by line.

In the second line a *FLOAT* array called Summand with three elements is created.

In the third and fourth line two more variables are defined. Afterwards the colours are chosen and the screen is cleared.

The seventh line is the beginning for a *FOR/NEXT* loop. First it prints the number of the summand that shall be entered. Afterwards the user must input this summand.

You see: It's possible to use variables as "array index". Before this program, we only used numbers.

Afterwards the summand will be added to the sum. It would also be possible to add all the three summands after the loop and assign them to the variable Sum, but think what would happen if there were far more than three summands – it would take a lot of time.

With the three summands we're using in this example the loss of speed isn't noticeable, but nevertheless we should not forget to optimize our programs. Later on we don't want to write only programs to add three summands, but also programs with which you can start, control and land space shuttles.

After the loop the calculation is displayed on screen, but that's clear anyway.

CHAPTER 8

File management

Sometimes you need to call external programs from our own program. The command for that is *SYSTEM*. The first parameter is the program (including path), the second one contains parameters. An example program would make no sense since it wouldn't do anything but start another program.

Let's better get to the actual file management. The most important command is *OPENFILE*. With it a file can be created or read. Its syntax is:

```
OPENFILE(FileVariable, Filename, Mode)
```

FileVariable is a variable of the type *FILE*. Using this variable the program accesses the file. *Filename* is the name of the file, including the path – e.g. "C:\TEMP\hello.txt". The *Mode* is either "R" (read), "W" (write) or "A" (append). While "W" creates a new file or overwrites an existing file, "A" opens an existing file and appends the new contents at its end.

Before the end of a program, the opened files must be closed again. For this, there's the *CLOSEFILE* command. Its syntax is:

```
CLOSEFILE FileVariable
```

Let's get to the reading of a file. This happens with the *READ* command. Syntax:

```
READ(FileVariable, Variable)
```

Actually *READ* isn't just a command but also a function. That means that *READ* returns a value. We can write:

```
a = READ(FileVariable, Variable)
```

a equals 0 if everything is okay. If *a*, however, has another value, an error has occurred. For instance the end of the file could have been reached. So we check the return value of *READ* in order to check whether the end of the file has been reached. If so, we stop.

Here's a practical working example, a text file viewer:

```
'File-Viewer, unuseful version
DIM File AS FILE
DIM FileName AS STRING
DIM Textline AS STRING
DIM a AS INT
COLOR 15, 1
CLS
PRINT "Filename"
COLOR 14, 1
LOCATE 1, 10
PRINT "?"
INPUT FileName
OPENFILE(File, FileName, "R")
CLS
COLOR 15, 1
'Read lines and print them
DO
 a = READ(File, Textline)
 PRINT Textline
UNTIL a <> 0
CLOSEFILE File
```

Why did I write 'unuseful version'? Simple: Try to open a large file!

The file is displayed without a break. You can effectively only watch the last line.

Let's include page-wise display. If the user wants to see the next page, he must simply press any key.

We can do that by including a counter in which we store how many lines have already been printed. Using *IF/THEN/ELSE/ENDIF* we check if it has reached a certain value (let's say 20). If so, the user must press a key.

Afterwards the screen is cleared, the counter is set to 0, and it goes on!

Here's the listing........... Why? It would be a good idea to leave its implementation to you as a home exercise! We've already talked about what the program should do.

Let's get to writing in a file. The writing command is *WRITE*. Syntax:

```
WRITE(FileVariable, Variable)
```

As an example, here's a simple ASCII texteditor.

```
'Texteditor
DIM File AS FILE
DIM FileName AS STRING
DIM Textline AS STRING
COLOR 15, 1
CLS
'Input Filename
PRINT "Enter the name of the file!"
PRINT "Attention: If a file with this name already exists, it will be overwritten!"
COLOR 14, 1
INPUT FileName
CLS
COLOR 15, 1
PRINT "Now enter the text! To end the program, enter an empty line."
COLOR 14, 1
'Input Text
OPENFILE(File, FileName, "W")
DO
 INPUT Textline
 WRITE(File, Textline)
UNTIL Textline = ""
CLOSEFILE File
```

That was all for today!

You've already got a home exercise. As always, you should also play with the new command. But watch out for "W"! If a file with the chosen name already exists, it will be overwritten with no mercy!

Solutions for Chapter 8

Exercise 1: You had to include page-wise display of the file. A possible solution is the following program:

```
'File-Viewer, useful version
DIM File AS FILE
DIM FileName AS STRING
DIM Textline AS STRING
DIM a AS INT
DIM b AS STRING
DIM Counter AS INT
```

```
COLOR 15, 1
CLS
PRINT "Filename"
COLOR 14, 1
LOCATE 1, 10
PRINT "?"
INPUT FileName
OPENFILE(File, FileName, "R")
CLS
COLOR 15, 1
'Read lines and print them
Counter = 0
DO
 a = READ(File, Textline)
 PRINT Textline
 Counter = Counter + 1
 IF Counter = 20
  INPUT b
  CLS
  Counter = 0
 ENDIF
UNTIL a <> 0
CLOSEFILE File
```

CHAPTER 9

Functions

Today we are talking about *SUB*s, one of the most important features of structurized programming. Before we can start with that, we must explain the difference between commands and functions.

Commands such as *PRINT* make the computer do a certain thing, e.g. output a text. You can attach parameters which make several options possible.

Functions such as *READ* can also use parameters. But functions have manifold uses. In contrast to a command, a function returns a value. Using this value you can do what you want: assign it to a variable, display it on screen, test it and so on. All of the following uses are valid:

```
a = READ(File, Textline)
PRINT READ(File, Textline)
IF READ(File, Textline) = "Y" THEN …
```

With the *SUB* command you can create your own commands and functions. Of course it isn't completely new commands, but only summaries of commands. You talk about a *program in the program* or a *sub-program*. This term is from the time in which GW-Basic and Basic 2.0 were supercool hits, but it's still used these days.

Let's create a simple *SUB* to see how it works!

```
'SUB-Demo
DECLARE Hello ()
DIM i AS INT

'Example program
FOR i = 1 TO 10
 Hello
NEXT i
END

'Actual SUB
SUB Hello
 PRINT "Hello world!" : 'A bit late, isn't it?
RETURN
```

Now I'll have to explain a lot. Let's go through the program line by line.

Line 2 (*DECLARE Hello ()*) is necessary so that the *SUB Hello* can be called like a command.

With the lines 6 to 8 we achieve that the *SUB Hello* is executed ten time in a row.

In line 12 the actual *SUB* starts. Behind the keyword *SUB* we must write the name of the *SUB*. In our example it's called *Hello*. The commands which are executed on calling this *SUB* are between the line with the command *SUB* and the line *RETURN*.

Bascially *SUB*s are nothing but blocks which can be called at any location inside the main program or inside another *SUB*. In an extreme case, a *SUB* can even contain a program of its own!

*SUB*s wouldn't be *SUB*s if they could not be called using parameters. To call a *SUB* with parameters, they must be defined in the line with *DECLARE*. Example:

```
DECLARE PrintText (Text:STRING, Number:INT)
```

Before somebody panicks, here's the explanation: When the *SUB PrintText* is called, two parameters must be written. With the above line we defined that the first parameter must have the type *STRING* and the second the type *INT*.

Text and *Number* are variable names using which the *SUB* accesses the two parameters. If we call the *SUB PrintText* with

```
PrintText ("Hello!", 1000)
```

the variable *Text* contains the value "*Hello!*" and *Number* contains the value *1000*. These variables can, however, only be accessed inside the *SUB*.

To make everybody understand it, here's an example program:

```
'Parameter-Demo
DECLARE PrintText (Text:STRING, Number:INT)

'Example programm
COLOR 15, 1
CLS
PrintText("Creative Basic is cool!", 20)
END

'Actual SUB
SUB PrintText (Text, Number)
 DIM i AS INT
 FOR i = 1 TO Number
  PRINT Text
 NEXT i
RETURN
```

If somebody still has problems with parameters: just ask!

Let's come to another important characteristic of *SUB*s, local variables. In the *SUB*s variables can be defined. They can be accessed only within this *SUB*. It's even possible to define variables that have the same name as variables from the main program.

Let an example program demonstrate it.

```
'Local variables
DECLARE ShowText ()
DIM Text AS STRING
COLOR 15, 1
CLS
Text = "See you!"
ShowText
COLOR 15, 1
PRINT "We are in the main program. The content of Text is:"
COLOR 14, 1
PRINT Text
END

SUB ShowText
 DIM Text AS STRING
 Text = "Hello!"
 COLOR 15, 1
 PRINT "We are in the SUB ShowText. The content of Text is:"
 COLOR 14, 1
 PRINT Text
RETURN
```

What are such local variables good for? Well, they have the following advantages:

• Every *SUB* has ist own local variales. Even if they have the same names as the variables from another *SUB* or from the main program, they are 'independent' variables.
• After quitting the *SUB* all the local variables are automatically deleted. If there weren't local variables, the programmer would have to do that.

*SUB*s can also return values. Thus they become real functions. A value is returned if you write it next to *RETURN*.

As an example I wrote a program using which you can compute the n-th root of a number. I'm using the '^' operator for that (power).

```
'Functions-Demo
DECLARE nthRoot (Number:DOUBLE, n:DOUBLE)
DIM Number AS DOUBLE
DIM n AS DOUBLE
COLOR 15, 1
CLS
PRINT "The n-th root of what number shall be computed?"
COLOR 14, 1
INPUT Number
PRINT
COLOR 15, 1
PRINT "How much is n?"
COLOR 14, 1
INPUT n
PRINT
COLOR 15, 1
IF n = 0
 PRINT "The 0-th root cannot be computed!"
ELSE
 PRINT "The n-th root of this number is:"
 COLOR 14, 1
 PRINT nthRoot(Number, n)
ENDIF
END

SUB nthRoot (Number, n)
 DIM Wurzel AS DOUBLE
 Wurzel = Number ^ (1 / n)
RETURN Wurzel
```

I don't want to explain the mathematical formula (it should be in every math encyclopedia).

CHAPTER 11

String processing functions

Now I want to introduce you to a couple of functions for string processing.

Function *ASC*. Syntax: *Variable = ASC(Character)*. The result is the ASCII code of the given character. Example:

```
'ASC-Demo
DIM Character AS STRING
COLOR 15, 1
CLS
PRINT "Enter a character!"
INPUT Character
IF LEN(Character) > 1
 PRINT "Error! You haven't entered a character, you've entered a text!"
 END
ENDIF
PRINT
PRINT "You've entered the character "
LOCATE 4, 30
COLOR 14, 1
PRINT Character
LOCATE 4, 31
COLOR 15, 1
PRINT ". Its ASCII code is "
LOCATE 4, 51
COLOR 14, 1
PRINT ASC(Character)
LOCATE 4, 53
COLOR 15, 1
PRINT "."
```

In this program also appears the function *LEN*. It returns the length of a string, that is is number of characters.

Function *CHR$*. Syntax: *Stringvariable = CHR$(ASCIICode)*. The result is the character with the given ASCII code. Example:

```
'CHR$-Demo
DIM ASCIICode AS INT
COLOR 15, 1
DO
 CLS
 COLOR 15, 1
 PRINT "Enter a number between 0 and 255!"
 COLOR 14, 1
 INPUT ASCIICode
UNTIL ASCIICode >= 0 & ASCIICode <= 255
COLOR 15, 1
PRINT "The character with this ASCII Code is "
LOCATE 3, 39
COLOR 14, 1
PRINT CHR$(ASCIICode)
LOCATE 3, 40
COLOR 15, 1
PRINT "."
```

CHR$(7) is an exception as it isn’t a character but a beep tone (simply try it!).

Function *STRING$.* Syntax: *Stringvariable = STRING$(Number, Character)*. The result is a string which consists of the given number of occurrences of the given character. Examplel:

```
'For people who are in love
COLOR 13, 0
CLS
FOR i = 1 TO 20
 PRINT STRING$(79, CHR$(3))
NEXT i
```

Function *MID$.* Syntax: *Stringvariable = MID$(String, x, y)*. The result is a partial string. It starts with the *x*-th character of the original string and has a length of *y* characters. Since this sounds more difficult than it is, here's an example:

```
'MID$-Demo
DIM StringVariable AS STRING
COLOR 15, 1
CLS
PRINT "Enter a string!"
COLOR 14, 1
INPUT StringVariable
PRINT
COLOR 15, 1
PRINT "The first six characters of the string are:"
COLOR 14, 1
PRINT MID$(StringVariable, 1, 6)
COLOR 15, 1
PRINT
PRINT "The characters 2 to 4 are:"
COLOR 14, 1
PRINT MID$(StringVariable, 2, 3)
```

See also: *LEFT$*, *RIGHT$*.

**Function *LTRIM$***. Syntax: *Stringvariable = LTRIM$(StringVariable)*. The result is a string which looks similar to the original *Stringvariable*, but without initial spaces if there were any. See also: *RTRIM$*.

Function *UCASE$.* Syntax: *Stringvariable = UCASE$(StringVariable)*. The result is that the lower-case letters have been converted to upper case. See also: *LCASE$*.

CHAPTER 11

The Creative Basic toolbox

I've written a little toolbox for you which contains a couple of *SUB*s that will make life easier for you. There are some command we haven't learned. If you want, search for them in *Help-User's Guide*.

```
'The Real Adok's Way to Creative Basic Library
DECLARE cPrint (Text:STRING)
DECLARE Formatted (Number:INT)
DECLARE nthRoot (Number:DOUBLE, n:DOUBLE)
DECLARE RandomNumber (Min:INT, Max:INT)

SUB cPrint (Text)
 PRINT STRING$(INT((80 - LEN(Text)) / 2 - 1), " ") + Text
RETURN

SUB Formatted (Number)
RETURN LTRIM$(RTRIM$(STR$(Number)))

SUB nthRoot (Number, n)
RETURN Number ^ (1 / n)

SUB RandomNumber (Min, Max)
RETURN INT(RND(1) * (1 + Max - Min) + Min)
```

A little documentation of the *SUB*s:

- *SUB cPrint*: The parameter is a string which will be printed on the screen centered.

- *SUB Formatted*: The parameter is an *INT* variable which will be converted to a string. Redundant spaces are cut off. This *SUB* is especially useful for printing numbers. It can also be adapted to support floating point variables.

- *SUB nthRoot*: The parameters are two variables of the *DOUBLE* type. We've already talked about this *SUB* in the previous chapter.

- *SUB RandomNumber*: The parameters are two integer variables. They determine the lower and upper limits of the random number that is to be created. The random number has the type *INT*. But this can be changed easily. If floating point variables shall be created, the call of the *INT* command must be modified.

Appendix

A couple of games in Creative Basic

Here there are a couple of games which can be crafted with the knowledge we gained in this tutorial.

***GAME0001* – Guess words:** This game is also known as "Hangman". The computer selects a random word from a database and displays only the first and the last character and the length of the word. Your task as players is to guess the word by entering characters. If you have guessed a character in the word correctly, it will be displayed, and you will be asked for the next character. If you have guessed wrongly, the error counter is increased by one. When you have made wrong guesses too often, you've lost.

```
DECLARE cPrint (Text:STRING)
DECLARE Formatted (Number:INT)
DECLARE RandomNumber (Min:INT, Max:INT)

DIM Word AS STRING
DIM WordDesc AS STRING
DIM DisplayWord AS STRING
DIM i AS INT
DIM DatabaseWord[20] AS STRING
DIM DatabaseDesc[20] AS STRING
DIM Ending As INT
DIM Character AS STRING
DIM Errorcounter AS INT
DIM CharacterFound AS INT

DatabaseWord[0] = "HOUSE"
DatabaseDesc[0] = "People live there."
DatabaseWord[1] = "ANIMAL"
DatabaseDesc[1] = "A creature."
DatabaseWord[2] = "SPOON"
DatabaseDesc[2] = "A thing you use for eating."
DatabaseWord[3] = "FORK"
DatabaseDesc[3] = "A thing you use for eating."
DatabaseWord[4] = "HEDGEHOG"
DatabaseDesc[4] = "A creature with spikes."
DatabaseWord[5] = "TEAR"
DatabaseDesc[5] = "A body fluid."
DatabaseWord[6] = "DINNER"
DatabaseDesc[6] = "Something to eat."
DatabaseWord[7] = "WINE"
DatabaseDesc[7] = "Something to drink."
DatabaseWord[8] = "KNIFE"
DatabaseDesc[8] = "A thing you use for eating."
DatabaseWord[9] = "TABLE"
DatabaseDesc[9] = "Furniture."

i = RandomNumber(0, 9)
Word = DatabaseWord[i]
WordDesc = DatabaseDesc[i]
DisplayWord = LEFT$(Word, 1) + STRING$(LEN(Word) - 2, ".") + RIGHT$(Word, 1)
CharacterFound = 2

DO
 DO
  CLS
  FOR i = 1 TO 9
   PRINT
  NEXT i
  cPrint DisplayWord
  PRINT
  PRINT WordDesc
  PRINT
  SELECT CharacterFound
  CASE 0
   PRINT "Too bad, it's wrong. Current state of error counter:" + STR$(Errorcounter) +
" of 5"
  CASE 1
   PRINT "Correct!"
  CASE 2
   PRINT
  ENDSELECT
  PRINT
  PRINT "Enter character and press ENTER:"
  INPUT Character
 UNTIL LEN(Character) = 1
 Character = UCASE$(Character)
 CharacterFound = 0
 FOR i = 1 TO LEN(Word)
  IF MID$(Word, i, 1) = Character
   CharacterFound = 1
    DisplayWord = LEFT$(DisplayWord, i - 1) + Character + MID$(DisplayWord, i + 1,
LEN(DisplayWord) - i)
  ENDIF
 NEXT i
 IF CharacterFound = 0 THEN Errorcounter = Errorcounter + 1
 IF DisplayWord = Word THEN Ending = 1
 IF Errorcounter = 5 THEN Ending = 1
UNTIL Ending = 1

CLS
FOR i = 1 TO 9
 PRINT
NEXT i
IF DisplayWord = Word
 cPrint DisplayWord
 PRINT
 PRINT WordDesc
 PRINT
 PRINT
 PRINT
 PRINT "Well done, it's solved!"
ELSE
 cPrint DisplayWord
 PRINT
 PRINT WordDesc
 PRINT
 PRINT
 PRINT "Too bad, it's wrong. Current state of error counter:" + STR$(Errorcounter) + "
of 5"
 PRINT "You've lost!"
ENDIF

END

SUB cPrint (Text)
 PRINT STRING$(INT((80 - LEN(Text)) / 2 - 1), " ") + Text
RETURN

SUB Formatted (Number)
RETURN LTRIM$(RTRIM$(STR$(Number)))

SUB RandomNumber (Min, Max)
RETURN INT(RND(1) * (1 + Max - Min) + Min)
```

Play a bit with this program. Extend the database.

***GAME0002* – Tic Tac Toe:** This game is for two players. One player draws crosses, the other player draws circles. The one who manages first to fill a row, a column or a diagonal with his/her symbol wins.

```
DECLARE DrawPlayarea()
DECLARE DrawCircle(x:INT, y:INT)
DECLARE DrawCross(x:INT, y:INT)
DECLARE Check(x:INT, y:INT, Player:INT)
DIM LeftTopX AS INT
DIM LeftTopY AS INT
DIM Window1 AS WINDOW
DIM PlayerWhoseTurn AS INT
DIM PlayareaStatus[3, 3] AS INT
DIM x AS INT
DIM y AS INT
DIM GameEnding AS INT
DIM NumberSetFields AS INT

LeftTopX = 10
LeftTopY = 10
PlayerWhoseTurn = 1

WINDOW Window1, 0, 0, 2 * LeftTopX + 310, 2 * LeftTopY + 250, @MINBOX + @MAXBOX +
@SIZE, 0, "Tic Tac Toe", Window1Prozedur
DrawPlayarea
WAITUNTIL Window1 = 0
END

SUB Window1Prozedur
 DIM Previous AS INT
 IF @CLASS = @IDCLOSEWINDOW
  CLOSEWINDOW Window1
 ENDIF
 IF @CLASS = @IDLBUTTONDN & Previous <> @IDLBUTTONDN & GameEnding = 0
  x = (@MOUSEX - LeftTopX) / 100
  y = (@MOUSEY - LeftTopY) / 75
  IF x >= 0 & x <= 2 & y >= 0 & y <= 2
   IF PlayareaStatus[x, y] = 0
    PlayareaStatus[x, y] = PlayerWhoseTurn
    SELECT PlayerWhoseTurn
    CASE 1
     DrawCircle(x, y)
     IF Check(x, y, 1) = 1
      MOVE Window1, LeftTopX + 55, LeftTopY + 105
      PRINT Window1, "Player 1 (Circle) has won!"
      GameEnding = 1
     ENDIF
     PlayerWhoseTurn = 2
    CASE 2
     DrawCross(x, y)
     IF Check(x, y, 2) = 1
      MOVE Window1, LeftTopX + 55, LeftTopY + 105
      PRINT Window1, "Player 2 (Cross) has won!"
      GameEnding = 1
     ENDIF
     PlayerWhoseTurn = 1
    ENDSELECT
    NumberSetFields = NumberSetFields + 1
    IF NumberSetFields = 9 & GameEnding = 0
      MOVE Window1, LeftTopX + 45, LeftTopY + 105
      PRINT Window1, "It's a draw!"
      GameEnding = 1
    ENDIF
   ENDIF
  ENDIF
 ENDIF
 Previous = @CLASS
RETURN

SUB DrawPlayarea
RECT Window1, LeftTopX, LeftTopY, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX + 100, LeftTopY, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX + 200, LeftTopY, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX, LeftTopY + 75, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX + 100, LeftTopY + 75, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX + 200, LeftTopY + 75, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX, LeftTopY + 150, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX + 100, LeftTopY + 150, 100, 75, RGB(255,0,0), RGB(255,255,0)
RECT Window1, LeftTopX + 200, LeftTopY + 150, 100, 75, RGB(255,0,0), RGB(255,255,0)
RETURN

SUB DrawCircle(x, y)
CIRCLE Window1, LeftTopX + x * 100 + 50, LeftTopY + y * 75 + 37, 30
RETURN

SUB DrawCross(x, y)
LINE Window1, LeftTopX + x * 100 + 10, LeftTopY + y * 75 + 10, LeftTopX + x * 100 + 90,
LeftTopY + y * 75 + 65
LINE Window1, LeftTopX + x * 100 + 10, LeftTopY + y * 75 + 65, LeftTopX + x * 100 + 90,
LeftTopY + y * 75 + 10
RETURN

SUB Check(x, y, Player)
 DIM i AS INT
 DIM ScoreCounter AS INT

 ScoreCounter = 0
 FOR i = 0 TO 2
  IF PlayareaStatus[i, y] = Player THEN ScoreCounter = ScoreCounter + 1
 NEXT i
 IF ScoreCounter = 3 THEN RETURN 1

 ScoreCounter = 0
 FOR i = 0 TO 2
  IF PlayareaStatus[x, i] = Player THEN ScoreCounter = ScoreCounter + 1
 NEXT i
 IF ScoreCounter = 3 THEN RETURN 1

 IF x = y
  ScoreCounter = 0
  FOR i = 0 TO 2
   IF PlayareaStatus[i, i] = Player THEN ScoreCounter = ScoreCounter + 1
  NEXT i
  IF ScoreCounter = 3 THEN RETURN 1
 ENDIF

 IF x + y = 2
  ScoreCounter = 0
  FOR i = 0 TO 2
   IF PlayareaStatus[2 - i, i] = Player THEN ScoreCounter = ScoreCounter + 1
  NEXT i
  IF ScoreCounter = 3 THEN RETURN 1
 ENDIF

RETURN 0
```

The game is mouse-controlled. With *IF @CLASS = @IDLBUTTONDN* the program checks if the left mouse button is pressed. The position of the mouse on the screen is read from the variables @*MOUSEX* and @*MOUSEY*.

Afterword

That was all folks! I hope the tutorial was fun and you learned something.

I didn't stick to a book when creating this tutorial. All ideas come from my head, and all the example programs were coded by me. This tutorial is the result of my long-time programming experiences. I hope, it was worth the effort and time I invested in this tutorial and it helped you get acquainted with the Creative Basic programming language.

I'm ready to help you with problems. You can already do quite a lot of things with the commands and functions you've learned in this tutorial. Everything else is up to your imagination, fantasy and ideas.

Learning a programming language opens a new world for you. Now you can communicate with the computer, make him understand your ideas and use its enormous potential for your own needs.

Everybody, no matter whether he is at school or on the job, is always confronted with various problems which can be simplified and solved by means of the computer. But don't only stick to doing routine jobs. It would be important to implement new ideas and create something new.

Once you've learned a programming language, you don't stop learning. There are no limits. Everything depends on you. Try to find new niches, take a risk, participate, don't be satisfied with what you've achieved!

Maybe we'll see each other again in another tutorial. I still have so many ideas…