Basic-Boss Compiler User Manual automatic translation done with Google Translator Toolkit.
Previous version: 30-July-2013
Actual version: 07-April-2015

##

BASIC-BOSS
Basic-Compiler

Source: 64'er Extra No. 11

Hans-Pinsel-Strasse 2, D-8013 Haar near Munich / West Germany

Table of contents

Foreword

Most had not expected that even a Basic compiler appears for the C64, which represents our past experience, in the shade. Now, it is still a reality: Basic-Boss translates your programs in pure machine language. Provide your programs even with so-called "compiler directives", so you can speed up your programs even further. This acceleration by a factor of "100" are not uncommon. You do not need in the future in many cases, the costly programming in machine language.

Many features allow a comfortable and performance-based work, because Basic-Boss contains, among other new commands and allows you to compile programs that were created with various Basic extensions. Moreover, the programmer can use a Basic-Boss parameter table ready to store frequently used default settings. We wish you much success with the Basic-Boss.
Your extra software editors.

Introduction

Once I had the idea to convert a basic program directly in machine language. Since the whole thing was a bit repetitive, I wanted to automate the process and sat in front of my C64 and programmed a Basic Compiler. After over a year of hard work and many sacrifices it was done: The Basic-Boss saw the light of the world. It (source code) now covers two floppy sides (41 tracks) and a complete run assembler requires about 30 minutes. The assembly is only on a 512-Kbyte RAM disk possible (I use Turbo Trans, but which has not some minor drawbacks. Thus, for example, it deletes every now and then uncalled individual sectors or entire disk sides). The assembler which makes such a huge project at all possible, is the "Assi" by Dirk Zabel. An equivalent approximately Assembler way I could not even distinguish yet on the "computer professional" Amiga and Atari ST.

Such extensive compiler also needs more detailed instructions. This consists of two chapters. Chapter "1" teaches the basics and introduces the operation of Basic-Boss. Most of you will be able to skip large parts of it, since there is a precaution all started from scratch and manages the user with almost no knowledge. But this section does not purport to be accurate or even complete, but is intended as an introduction. Nevertheless, one can already work with the Basic-Boss after its reading. In contrast to chapter "1" offers chapter "2" an accurate and complete description of all commands and characteristics of the Basic-Boss. But there is usually not provided much knowledge.

In order to understand the manual and can use the compiler, however, some basic knowledge of the Basic programming language are necessary because only the features and commands of the compiler are described, but not that of Commodore Basic. Therefore, the basic beginner to recommend a good basic book, because the standard Commodore manual is pretty thin.

The entire manual was written by the way on the C64 VizaWrite. I prefer the C64 in word processing over the Amiga and the Atari ST (which are next to useless) and consider the statement to be absurd, the C64 was too small for reasonable word processing. Before working with the compiler, you should make a backup. This is not a problem as the Basic-Boss contains no copy protection on the one and the other can even copy itself. And it goes like this: Make sure that you have a largely empty and formatted diskette available and then download the Basic-Boss of the original disk with

```
LOAD "BASIC-BOSS",8
```

When the computer has finished loading, type "RUN". Now you can push your formatted floppy disk in the drive. Then press the <*> button and enter a name under which the Basic-Boss is then saved when you exit the entry with <RETURN>. On the original disk are still a few example programs that you should look at. Otherwise, I hope you enjoy reading this guide and I hope that you do not have too much trouble with the Basic-Boss.

- Chapter 1 - Introduction -

1.1 Basics

1.1.1 Basic and machine language

The C64/C128 includes as a factory default, the Basic programming language. This has a reason. Basic is an easy-to-learn language to recommend to any beginner. But Basic not only work with beginners. Also, advanced and professionals prefer to write their programs in Basic as in any other language, because Basic has powerful mathematical functions and a comfortable string management and certainly allowed the development of larger programs. Basic could easily prevail against it only serious competitor. This competitor is the machine language. It has no string management, can not count on floating-point numbers and is generally very primitive and extremely cumbersome. In addition, it takes a long time for large programs until you can test out the writing at all. In Basic, the entry of "RUN" will suffice. But despite these drawbacks the machine language is often used because it has a distinct advantage: it is much faster than Basic.

This difference is due to the different principles of the two languages. The machine language is interpreted by the computer itself (the machine). Why it is so fast and so primitive, because it would be too expensive to build a computer that understands a less primitive, high-level language. Since the user can not expect such a primitive language, to write in this machine language program that processes a higher or interpreted language. This is the interpreter. Now these can be programmed in high level language, for example in Basic. Their commands are recognized and executed by the interpreter. However, this process requires a lot of time, and therefore Basic is just so slow.

It seems impossible to use the advantages of programming in Basic, if you want to write fast programs, because for most applications Basic is unbearably slow. But there is a way out of this dilemma: one can translate someone's own Basic programs into machine language. There will be for this are anything but boring job, it makes sense to automate this process and to leave it to the computer, repetitive tasks to accomplish because it is there eventually. The only issue is that it is necessary a translator program (the compiler), because is enormously expensive and complicated, since the task is in fact not monotonous. That is why many compilers confined to implementing the Basic program into a more efficient something special code that is interpreted but then again, why only a moderate acceleration is achieved.

The first real compiler was published in the magazine 64'er long time ago As-Compiler that actually produced real machine code. However, he understood only a very limited basic and was therefore only very limited use. However, the Basic-Boss understands all the basic commands and even creating a more efficient code. But you should be quite clear that even the Basic-Boss does not need the machine language, because very time-critical programs will be written as before in machine language, since a compiler can never generate efficient machine programs, like a real person can write. But now you can with the Basic at least an order of magnitude to reach the speed of the machine language.

1.1.2 Data Types

The low speed of the BASIC interpreter does not result solely from the interpreter principle. Another is the exclusive use of floating point numbers (for example, 1.2345 or 572545.2647254). Such figures, the processor of the C64 (which is responsible for all calculations and operations block in C64) process only with great difficulty, which is why it is quite slow. Therefore it is natural to use simpler numbers, the processor can process easier. These are integers, so completely normal numbers without decimal places. If you look at Basic programs, one will realize that no decimal places are required in most cases

and you could use instead of floating-point variables, integer variables. Such variables are actually. Every now and then you come in Basic programs to variables of the form "A%". These variables can only accept integers from -32768 to +32767. One might suspect that these numbers are processed faster therefore. The statement A%=B%+C% should be faster than A=B+C. However this is not so. These so-called integer numbers are even slower, which is not located on the figures themselves, but to the lack of efficient implementation in operating system.

After all, the Commodore Basic has thus been called three data types or in other words: Basic provides three ways to display data. This is firstly the representation as Floating Point Numbers, also called Real (for example, A = 1.2345), on the other hand, the representation as Integers (A% = 10000) and also the representation as Strings, the strings are called (for example, A$="cuckoo"). Thus, in the three basic types available data "Real", "Integer" and "String".

But two of these data types are almost completely unsuitable for the processor (Real and String) and the third only partially (Integer). For the Integer type is often useless when working with memory addresses, so with PEEK and POKE, because the memory cells of the C64 are known numbered from 0 to 65535 and not -32768 to +32767 (Chapter "2" is explained, why both are actually the same). Therefore it is natural to introduce more data types, where the deficiency is not attached. I call these types "Word" and "Byte". A variable of type "Word" is an integer from "0" to "65535," and is therefore well adapted to the C64 and its processor. Even better fits the type Byte to the processor, because with the 6510 (the processor of the C64) fits exactly into one of the 65536 memory locations of the C64. A variable of this type can assume an integer from 0 to 255. If these types are used in the basic program, you can translate it into a very fast machine program.

1.2 Directives and compiler operation

It raises the question of how to use these new data types at all, because the Basic interpreter does not understand it so, and you have to somehow tell the Basic-Boss that you want to use it. For example, you wrote the following program:

```
10 for i=1024 to 2023
20 poke i,160: next i
```

Of all 65536 memory locations, which has the C64, the program selects the which lie between the 1024th and 2023th and fill it with the value 160. But because the screen memory is right there, the screen is slow (in some C64 to complete only the lines of text, but that should not bother) from top to bottom filled. Since you would like it a little faster and ragged, you will compile the program, for example, with the Basic-Boss.

And that's what you can do now. So turn on your computer, monitor and drive and write this little Basic program. If you start it with "RUN", you can surprise yourself of the very high speed. Then you should save it in a not-too-full disk under the name "TEST". Then push the Basic-Boss disk into the drive and load the Basic-Boss:

```
LOAD "BASIC-BOSS",8
```

Figure 1: How the Basic-Boss reports after the start

Now it can take something without floppy accelerator (which is why I would advise such a principle-tion. On the Basic-Boss diskette, see the speeder software "Ultraload Plus" which, although a hardware speeder can not replace, but still represents a big help. Please refer to Appendix E). When "READY". appears type "RUN". Now if a page of text is displayed, so you can make this disappear with a function key (<F1> <F3> <F5> or <F7>). Now you are prompted for the source file, after which you enter "TEST", after you have inserted the disk on which the test program is located. If you have the entry with <RETURN> completed, the Basic-Boss begins its work. He reads the BASIC program from disk and creates a machine program, which it stores under the name "+TEST" (this process is called "compile"). If an error occurs in between, he interrupts his work and waits for the operation of <SHIFT>. Then you need to improve the errors and start from scratch.

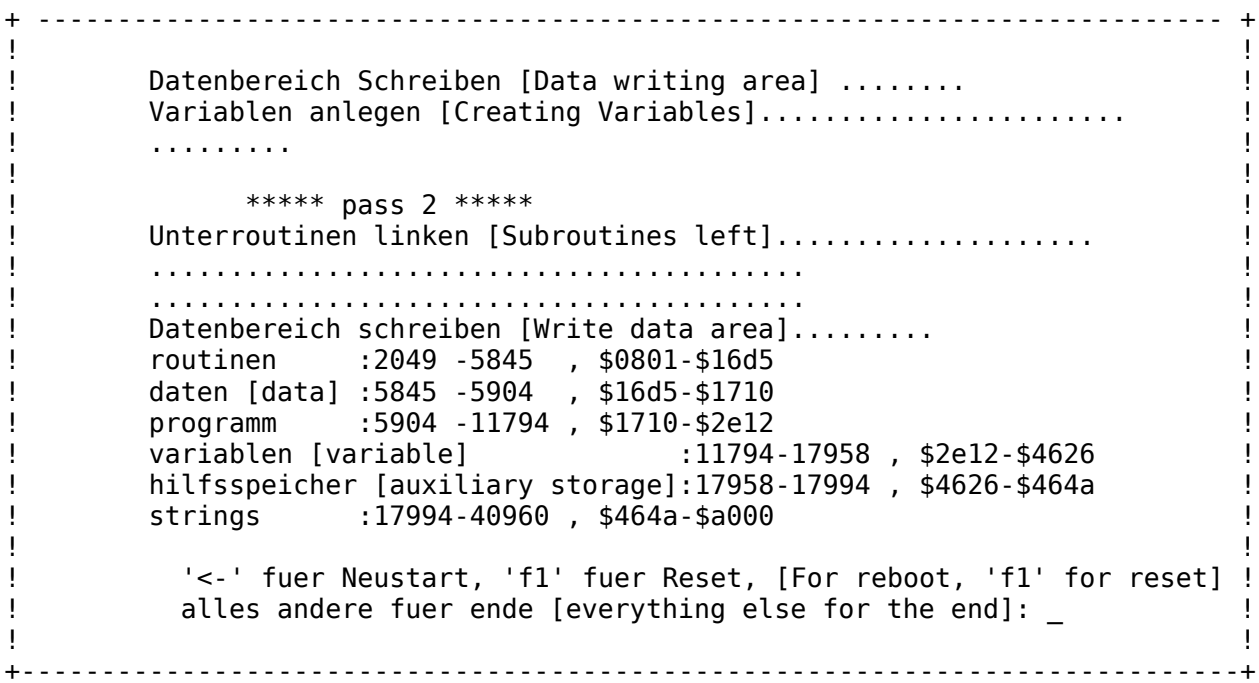

Figure 2: A program has been successfully compiled

The finished machine program (also "compiled code" called) can normally load with

LOAD "+TEST",8

and started with "RUN". What you see now, is indeed much faster than the basic program, but you can probably still not quite convinced that the Basic-Boss should be worth it. Because if you use any other compiler, the result will be about as fast. You want to know why? The

answer is simple: it was, as usual, expected real variables and that takes times now its time. But there are really only integers between 0 and 65535 are used in the program, the data Word type would be much more meaningful. But how to share this with the Basic-Boss? The Basic interpreter distinguishes its data types (Real, String, Integer) based on a double-digit appended to the variable name special character. In real variables nothing is appended (for example, "A"), a String variables "$" appended (for example, "A$") and integer variables is a "%" appended (for example, "A% "). The Basic-Boss makes basically the same. But it can also communicate in other ways, of which type is to be a variable. For example, if the variable "I" of the example program type should be Word, then it must in the first line of the program, the command "£WORD I" are, like so:

```
5 £word i
```

This command is a compiler directive. It tells the compiler how it has anything to do (he has the variable "I" to be regarded as a Word variable). Such a directive always starts with the "£" sign. The compiler would accept the program that way. The interpreter but complains when entering "RUN". And rightly so, because "£WORD" he does not understand. Therefore it is useful to have a "REM" to write before the command. Then the interpreter accepts the program and executes it properly - the Basic-Boss ignores the REM line, however. Thus, this dilemma can be resolved, there is a way to make ineffective a REM command for the Basic-Boss. Just use:

```
5 rem@ £word i
```

Then the Basic-Boss ignores the REM command line and evaluates it. The interpreter, however recognize the REM command and ignored about the rest. When you now load your test program, add it to this line, save them back to disk and compile, then you are at the machine program ("+TEST") find a strong increase in speed (this is now roughly the speed of "Petspeed" with integer variables).

1.2.1 Other compiler directives

But it goes even faster. The FOR-NEXT loop of the Basic-Boss is quite cumbersome and slow just because it is unfortunately no other way, if it is to do exactly the same as that of the interpreter. However, you can instruct the Basic-Boss, to make it better. This is done with the command "£FASTFOR". However there are a few limitations that you should look in the chapter "2" under "£FASTFOR" and "£SLOWFOR".

If you rewrite the test program as follows:

```
5 rem@ £word i: £fastfor
10 for i=1024 to 2023
20 poke i,160: next i
```

and compile, then no distinction can be established to the machine language of a purely visual. Apart "£FASTFOR" particularly the compiler directives "£ALLRAM" and "£OTHERON" are important. "£ALLRAM" instructs the Basic-Boss to use the entire memory of the C64 for the basic program and the variables. If "£OTHERON" is at program start, then are allowed by commas parameters attached to the SYS command, for example. Both commands must, as usual, are on the program start. For more information you should look up in chapter "2".

1.2.2 Examples

To give you an idea of the application of the new data types and the compiler directives, I'm showing a few small examples. They will be compiled and executed as described.

```
Example 1:  10 rem@ £byte a
            20 a=0
            30 poke 53280,a: a=a+1
```

```
            40 if a>15 then a=0
            50 goto 30
```

The color screen is switched through with all 16 colors. Because the variable "A" can only have values from 0 to 15, it can be declared as a Byte variable.

```
Example 2:  10 rem@ £byte a(,x: £word #
            20 dim a(20)
            30 sum=0: mul=1
            40 for x=1 to 20
            50 input a(x)
            60 sum=sum+a(x): mul=mul*a(x): next x
            70 print sum, mul
```

In this example, via "INPUT" read 20 Byte values in the array "A(" and calculates their sum and multiply. Something seems strange to line 10 you can see how arrays (fields) are defined. The variable name is simply a bracket is appended. Interesting is also the command "£WORD #". A variable "#" does not exist and cannot exist. With this command, all variables without additive (for example "A", "AB" and "COL", etc. and not "A%", "B$", etc.) will be declared Word if it is not already on elsewhere declared with a variable type. Variables of the form "A%", "VOL%" and so on are declared with "£WORD %" as a Word. So now are "SUM" and "MUL" Word type and "A(" and "X" of type Byte.

```
Example 3:  10 rem@ £allram: £fastfor: £word i: £byte a(
            20 dim a(62000)
            30 for i=0 to 62000
            40 a(i)=100
            50 next i
```

It tells the compiler by "£ALLRAM" that he will use all the memory. Then, a byte array is dimensioned to 62000 elements, filled at 100.

Well you have come to the end of the first chapter. You can now use the compiler and use. A detailed description of the Basic-Boss, its commands and its properties, see part "2" of the manual.

- Chapter 2 - Operating Manual -

Here you will find an accurate, complete and systematic description of all the features and commands of the Basic-Boss. Therefore, this chapter is very well suited for future reference.

2.1 Operation of the Basic-Boss

2.1.1 The contents of the Basic-Boss diskette

```
0 38745 basic-boss m&t
193 "basic-boss"        prg  main program
0   "---------------"  del
9   "lies mich"         prg  information about disk
0   "---------------"  del
4   "other"             prg  example of £OTHERON and £OTHER (Basic)
12  "+other"            prg  example of £OTHERON and £OTHER (compiled code)
0   "---------------"  del
1   "newcom.o"          prg  Basic extension for !COL (start with "SYS 49152")
3   "newcom.src"        prg  source code for Basic extension (as a text file)
0   "---------------"  del
3   "errorread"         prg  ASCII file loader Basic (only run as compiled code)
13  "+errorread"        prg  ASCII file loader (compilated code)
0   "---------------"  del
5   "preferences"       prg  Example for default settings
0   "---------------"  del
3   "bossdatei"         prg  Demonstrates the new file commands (Basic)
7   "+bossdatei"        prg  Demonstrates the new file commands (compiled code)
0   "---------------"  del
1   "bossdir"           prg  Shows Directory of a Disk (Basic)
5   "+bossdir"          prg  Shows Directory of a Disk (compiled code)
0   "---------------"  del
3   "ramrom"            prg  Copies the graphics memory from ROM and return (Basic)
6   "+ramrom"           prg  Copies the graphics memory from ROM and return (compiled code)
0   "---------------"  del
3   "ramload"           prg  Loading a file into RAM (Basic)
13  "+ramload"          prg  Loading a file into RAM (compiled code)
0   "---------------"  del
2   "ramdir"            prg  Source code "r+ramdir" and "p+ramdir"
5   "r+ramdir"          prg  Demonstrates £ROUTSTART
1   "p+ramdir"          prg  Demonstrates £PROGSTART
0   "---------------"  del
8   "autoboss"          prg  Demonstrates the automatic function
3   "auto.short"        prg  Short version to embed in your own programs
0   "---------------"  del
8   "forreturn"         prg  Demonstrates FOR and RETURN used incorrectly (Basic)
14  "+forreturn"        prg  Demonstrates FOR and RETURN used incorrectly (compiled code)
0   "---------------"  del
7   "fastmuldiv"        prg  Fast multiplication and division (Basic)
14  "+fastmuldiv"       prg  Fast multiplication and division (compiled code)
0   "---------------"  del
12  "break"             prg  Demonstrates great way to increase the speed (Basic)
27  "+break"            prg  Demonstrates great way to increase the speed (compiled code)
0   "---------------"  del
28  "demo"              prg  Demo screen output (Basic)
38  "+demo"             prg  Demo screen output (compiled code)
0   "---------------"  del
27  "labyrinth"         prg  Game (Basic)
59  "+labyrinth"        prg  Game (compiled code)
0   "---------------"  del
1   "fastdir"           prg  Fast Directory display (Basic)
11  "+fastdir"          prg  Fast Directory display (compiled code)
0   "---------------"  del
1   "bigarray"          prg  Demonstrates £ALLRAM (Basic, only compiled executable)
4   "+bigarray"         prg  Demonstrates £ALLRAM (compiled code)
0   "---------------"  del
37  "wayout"            prg  Game
0   "---------------"  del
9   "ultraload plus"    prg  Quick Loader
```

```
1   "ultraload tool 1"  prg
2   "ultraload tool 2"  prg
0   "----------------"  del
74 blocks free.
```

2.1.2 Operation of the compiler

The compiler load with:

```
LOAD "BASIC-BOSS",8
```

and started with "RUN". Now if a page of text on the screen, you can have this with a function key (<F1> <F3> <F5> <F7>) disappear (but this is not essential). Then a copyright message and version number appears and the user is prompted for the source file. Now you can display the directory with "$" + <RETURN> or immediately enter the file name (with the naked pressing <RETURN> it is accepted with "BASIC"). Then start the compilation. This consists of two passes (Pass 1 and Pass 2). During pass 1 is read the source file and performed the distribution of addresses. In pass 2, the Basic program is again read from the machine program that can be generated, which is simultaneously written to the disk. The name of the generated machine program is by default generated from the source file by adding a "+" at the beginning. If an error occurs during compilation, the number of the faulty line is displayed. After the number is an extract of the previously processed part of the current line to find that the error can be more easily located. This extract consists of double points, arithmetic operators, and one point for each command. Below is the actual error message after an error number to facilitate (Chapter 2.12.1) to locate the error in the error list. The compiler stops on any error inherent to this not being seen again disappears from the screen (unless it was the "£PROTOCOL" command is used). Pressing the <SHIFT> finished this state of pause. The error messages are displayed in both passes and therefore often appear twice, but that did not bother. At the end of pass 1 Basic-Boss report the not dimensioned arrays, which are then dimensioned to 10. You can always abort the compilation with pressing <RUN/STOP> with <RESTORE>. After completion of the compilation the Basic-Boss shows the addresses of the various areas and prompts the user, either to press < <- >, <F1> or any other button . With < <- > the compiler is restarted, <F1> performs a reset. Any other key stops the Basic-Boss without reset. The Basic-Boss also offers another memory function, with which you can save him. If the name you require is a star when entering the source file ("*"), then the compiler under this name is written on drive "8". The At sign can be applied here by the way safe (for example, "* @:NEW BOSS").

2.1.3 Automatic Mode

If you often compiled, for example, if a program can only be run as compiled code, then you will often type in always the same. It would be better if you could automate the operation of the Basic-Boss. Such a possibility there is in fact. One can convey various information to the compiler. You are in the cartridge buffer from 900 memory location. When run, the Basic-Boss first examined the locations 1000 and 1001. He finds the values 123 and 234, it is assumed that the user wants to automatic mode. From 1002 he then gets the device number for the source file, from 1003 the corresponding secondary address. In 1004, he expects the length of the file name and the file name itself from 1005 The program described in this way is then compiled. Once the compilation is complete, the Basic-Boss picks up a device number of 900, a secondary address from 901, a name length of 902 and a name from 903. Then this causes that the reloaded program starts and stops itself. The program can for example, be reloaded the compiled program, which can then be tested immediately.

However, it is not as simple, but much more complicated to tell the Basic-Boss in this way the name of the source file, since it requires many POKEs. Thus, one should use for the POKEs a small program that uses the locations from 900 correctly and then the Basic-Boss invites. You can save even more work when this little program integrated into the to be compiled Basic program, it expanded to a few floppy commands and a SAVE command and press "£IGNORE" and "£USE" clings to it the compiled not meaningless extended. Then you need it

only with "RUN start line" call to automatically save the source program and to allow compiling, wherein the compiled program will also automatically loaded and started. A sample program did illustrates the procedure is to be found on the program disk under the name "AUTOBOSS" The program is based on a small basic routine that isolated and packaged into "AUTO.SHORT" is and can therefore be easily incorporated into your own programs.

2.2. General

2.2.1 Compiler directives

The Basic-Boss is controlled almost exclusively by means of compiler directives. These are commands that are like others in the Basic program. But they do not meet any normal function such as "PRINT" or "POKE" but tell the compiler what he has to do. All Compiler directives begin with the pound sign (£), based on the C64/C128 keyboard with a single button (button between is accessible <-> and <HOME>. Then the command name and then possibly some parameters follow.

2.2.2 Number systems

Numbers accepted the Basic-Boss both in decimal and in hexadecimal form, with hexadecimal values with a dollar sign ("$") are introduced. In addition, it can be any length and you don't need to take on consideration the randomly generated Basic Commands. For example, the following is possible: "PRINT $F, $DEF, $123456789ABCDEF" or "GOTO $1E". Decimal numbers can only be specified in decimal form.

2.2.3 Inactivation of REM commands

The Basic-Boss reacts to a REM command as well as the interpreter: he ignores the rest of the line. This can be prevented, however, if you write directly behind the REM command an At sign (@). Then the compiler examines the lines available after the REM. After REM one can only write compiler directives and compiler commands due to a peculiarity of the interpreter (tokenization) but Basic commands are not recognized there. The disabling of REM commands is useful to hide compiler directives before the interpreter, so that this does not complain when debugging a Basic program with an error message if it contacts a directive.

2.2.4 The compilation

The compilation is normally provided by the Basic-Boss with a SYS-line stored as a normal program. It can be like a BASIC program loaded and started. But you can also put in a specific memory area and a Basic program to call out with "SYS start address" (see chapter 2.4.3 "£ROUTSTART"). The Basic program works after the compilation more properly, as all are important locations for Basic restored after the compilation again.

2.3 Data Types

The type system is probably the most important aspect of the Basic-Boss. It has the types: Byte, Integer, Word, Real and String. There is also the type Boolean, which is however quite important and is treated in most cases as Byte. Every variable in the Basic program can be one of these types.

2.3.1 The range of values of types

Real and String are the most used in Basic data types. Their range of values likely to be known to you. While one can speak little of the String type value range, as it describes strings of (almost) arbitrary length, can be better the Real type samples. It describes

decimal floating point numbers in the range of + -2.93873588 * 10(-39) to + -1.70141183 * 10(38). When Integer type, the value range is considerably easier from: It ranges from -32768 to +32767. Word summarizes values from 0 to 65535 (= (2 to the 16)-1) and byte from 0 to 255 (= (2 to the 8)-1). The types Byte, Word, Integer and Real values are chosen for their range called numerically. Boolean can really only take the values "0" and "-1". This type is needed for decisions.

2.3.2 The declaration

The interpreter recognizes the data type of a variable on the variable name added. If no additional follows, for him is the variable of type real (floating point). If "%" followed, he concludes on an integer variable (for example, "A%") and "$" to a string variable (for example, "B$"). The Basic-Boss holds by default also follow this rule. One can assign to variables but also other types. This is done with a compiler directive. This generally looks like this:
£Type variable list

Instead of "Type" of each of these types can be (Boolean, Byte, Integer, Word, Real, String). The variable list consists of one or more variable names separated by commas. An example:

```
10 £byte a,b%,c
20 £word co
```

Here are the variables "A", "B%" and "C" for Byte variables and the variable "CO" for Word variable. You can declare the variables according to their additions as well. If you want, for example, all the variables of the form "A%" declare as a Word variable, then it is sufficient to take the potentially large number of variable names to use in the simple declaration "%". Consequently, you can also declare the variables of the form "A$" differently, by using "$". Particularly important is the possibility to assign a specific type to variables without additive. This is possible with the hash character ("#"). The declaration of variables by means of its name, however, has priority over the declaration on the basis of the additive. One can therefore define as Word and still assign the type Byte of the variable "X%", for example, all the variables with the suffix "%". The order of declarations is basically irrelevant.

Example: 10 £byte #,x%: £word a,b,%,c

Are declared with the addition of "%" as a Word, the variables "A", "B" and "C" and all. All variables and without the addition of the variable "X%" is assigned the Byte type. Since the explicit declaration takes precedence, is "X%" of type Byte and not the Word type, as well as "A", "B" and "C" are Word type and are not of type Byte. The high flexibility of course you can also abuse:

```
10 rem@ £string #,a%: £word $: £real %
20 x$=2: y$=3
30 a="hallo": a%="lollipop"
40 b%=1.23456
50 print mid$(a,x$,y$),a%,b%
```

The interpreter gets off with a "Type Mismatch" but the compiler absorbs the program without difficulty.

2.3.3 FAST variables

In the programming language "C" on the Amiga or Atari ST, there are register variables, which tries to store the C compiler in the registers of the main processor so that they can be processed very quickly. When the processor C64/C128 this would be a hopeless task, since it has only a few very small tab that why he constantly needed elsewhere. But he has the

nice property that it (the so-called Zero Page) processed very efficiently the lowest 256 bytes of its address range. Therefore allowed the Basic-Boss, a few variables of type Byte, Word or Integer (no arrays) to be placed in this area. In normal operations bring such a variable speed gain of about 25%. The memory consumption decreases by about a third. Dramatic is FAST variables affect in PEEK and POKE. If such a variable is used instead of a normal variable for the address, then these operations are accelerated by a factor of three and require only one-third of the program memory. Therefore one should FAST variables apply especially here. When declaring a FAST variables the variable name with an equal sign is simply appended to the "FAST" follows (for example, "£WORD A=FAST, B=FAST"). However, it should be noted that only four bytes for FAST variables are available. It can therefore only two word variables are declared with FAST or four byte variables or word variable and two byte variables. The addresses are used 251 to 254. Note: Fast start of the program variables are not like everyone else set to "0". Therefore, you need to be initialized by the programmer.

```
+-------------------------------------------------------------+
!                                                             !
!                                                             !
!          *** Basic-Boss Compiler ***                        !
!              Version 2.40                                   !
!             (C) 1988 by Thilo Herrmann                      !
!                                                             !
!          Quelldatei: [Source File]                          !
!          test 57                                            !
!                                                             !
!          20 .,,                                             !
!          62/zuviel 'fast'-variablen [too much 'fast' variables]!
!                                                             !
!                                                             !
+-------------------------------------------------------------+
```
Figure 3: Error Message 62: There were too many FAST-declared variables

2.3.4 Address establishing with variable

Normally, the Basic-Boss determines the address of each variable itself. However, you can specify the address of a variable, provided it is numeric. This is done similar to the FAST declaration must only instead of "FAST" an address to be specified, for example: "£WORD A=49152". Arrays can also be declared ("£BYTE SCREEN=1024"). Now you can for example very easily appeal to the tab for the background color, if you declare "COL" as follows: "£BYTE COL=53281". You can also speak to any variables of machine programs by placing their own variables to the affected locations. Another application is the transfer of values to programs that are loaded and include the same declarations. You can also create more FAST variables by placing normal variables on the zero page address of $2B to $2E, which otherwise are not touched by the compiled code. In fact, the statement "£WORD A=$FB" respect "A" is identical with "£WORD A=FAST". Just as FAST-defined variables are variables to a specific address is not initialized and the program beginning any random value.

2.3.5 Advantages and disadvantages of the types of data and use the correct type

One may ask, what for sure five types of data are needed. The answer is simple: use the correct data types increases the efficiency of a program. Thus, it is faster and shorter. Essential to the decision of the correct data type is the desired range of values. In addition, it is preferable to always use the data type that consumes the least memory and is the fastest. The following applies:

- Byte is the fastest and uses the least memory variables (just one byte)
  and program memory.
- Word is about half as fast and consumes exactly twice as much memory variables
  (Two bytes), and usually more than twice the amount of program memory.

- Integer is just as fast as Word for many operations and uses as
  much memory variables. For integer comparison operations but is much slower
  (Except for "=" and "<>").
- Real is much slower than the types described above and consumed whole
  five bytes of memory variables, but due to a special calculation method somewhat
  less program memory than Word.
- String is an extremely complex data type and therefore not too fast.
  But the editing is still a lot faster than the Basic interpreter, not only
  what garbage collection is concerned. The space requirement is similar to the Real type.
  Added to this is the space requirement of the text string.

It follows from this that you should use the Byte type, if possible, otherwise Word, otherwise Integer and only if it is absolutely necessary Real. If you use for example the variable "I" exclusively for string processing in the form of "A$=LEFT$(B$,I)", then you should declare "I" as a Byte variable, since "I" only values between "0" and "255" are meaningful. If a variable "X" will still be used in the form "POKE X,0", it should be defined as a Word. For more information, to improve the efficiency of programs, see "Efficient Programs (Section 2.8)". The expected types in commands and functions you can read under "types in operations, functions, and commands" (Chapter 2.3.9).

2.3.6 Conversions of datatypes

Because different data types may conflict with each other, they must be converted into each other every now and then. One can differentiate between two types of conversion.

2.3.6.1 Conversion with loss

Take a look at the following program:

```
10 rem@ £real a: £byte b
20 a=97.374546
30 b=a
40 print a,b
```

Under this program, the interpreter is running properly. There is indeed only expected Real variables. But when compiler it would be a problem, because in line 30 is a Real variable assigned to a Byte variable. This is actually technically impossible, because you can not assign a five-byte and very accurate real value of a primitive Byte variable, but the only whole numbers between "0" and "255" interpreted. But the Basic-Boss compiled this program without any difficulties. If you start it, you can see the principle: there are simply cut off all decimal places and of 97.374546 is simply 97. However, real numbers may be larger. If a Byte variable is now assigned as 258.123, cutting off the decimal places is not enough. In addition, the number is then made simply smaller by basically only the remainder of a division is made by 256. Easily explained: it is so often taken 256 until the result fits into the Byte variable. In this case, the Byte variable would after assigning the value "2" included (258-256). It is similar with the other transformations. If Real is to be converted to Word or Integer, the decimal places are also cut off and taken only the rest of the division by 65536. This is somewhat surprising for Integers, because this type has a different range of values, but more on that later. If Word or Integer to be converted to Bytes, only the remainder of the division is again taken over by 256 (in other words: It is simply the high byte omitted).

2.3.6.2 Conversions without loss

In the transformations described so far always been something lost or the accuracy decreased. However, there are a number of other transformations, which do not have this disadvantage. These are conversions of Bytes to Word, from Byte to Integer or from Word to

Real. In these cases, the value range of the source type is always included completely in the value range of the target type. The transformation of Boolean way, is also lossless.

2.3.7 Value of types and their processing

```
10 rem@ £word w: £real r
20 r=1.5: w=4
30 print r*w
```

When this program is run under the interpreter with "RUN" then the result of "6" appears on the screen. Whatever else in this example, the compiler but comes back to a type mismatch. In line 30, the real variable "R" is to be multiplied by the Word variable "W". The Basic-Boss can not associate variables of different types. The multiplication can be performed, consequently only when one of the variables has been converted into the other type. So, for example, the compiler can convert the content of "R" in the Word type to multiply then. He would convert the decimal value 1.5 into a Word value and therefore cut off the decimals. He then would receive 1. Then he would multiply this "1" with "4". The result would be "4" and obviously wrong. It should be better to convert the contents of Word variable "W" into a real number. Then the Word value "4" convert to the real value of "4" and would result with "1.5" multiplies the correct result "6". This is exactly what the Basic-Boss do. If it has a choice, it always converts the type from the smaller range of values to the larger range. This, he decides, based on the value of some type. The lowest value of Boolean and Byte. Then follows Integer, before Word and before Real. In one type of conflict, the Basic-Boss preferably opt for the type with the higher value. Meanwhile, you should always be aware when you provoke a type mismatch. Another problem:

```
10 rem@ £word wl,w2: £real r
20 w1=3: w2=2
30 r=w1/w2: print r,w1/w2
```

In line 30 the result of the W1/W2 real variable "R" should be assigned. "W1" and "W2" are both of type Word, so the division could be done in just this type. However, the result of W1 by W2 would not be 1.5, but 1 was not expected because real numbers. This incorrect result would then be assigned to "R". To prevent this, the Basic-Boss has basics assignments minimal with the target type, in this case Real. Therefore, also agrees with the first output from the PRINT command in line 30 value. The second is true because the compiled code of the indicated problem is not entirely because there is no target variable in the PRINT command (which is why the minimum type for PRINT with "£PRINTTYPE" can be set).

2.3.8 The diffuse range of Byte, Word and Integer

2.3.8.1 Byte and Word

It has been claimed that a Byte variable can only assume values between 0 and 255. But it looks like this:

```
10 rem@ £byte a,b
20 a=5: b=-1
30 a=a+b
40 print a,b
```

To the value of the variable "A" (5) the value of "B" (-1) is added. The result would be 4 and, behold, it works. "A" is "4" displayed by the compiled program with. Particularly interesting is the content of "B", because it can not be -1 because it does not fit into the value range. "B" is displayed in line 40 with "255". But why 255? -1 is less than 0 So you have to go back a step from 0. To take another example, if 10 is reduced by 1, we obtain 9 Suppose someone sees only the rearmost position and knows why only the numbers from 0 to 9 When he observed the number 10, he only sees the 0, He was told that it would now subtract 1 and suddenly out 0 become a 9. As the number at the end of the value range was known to him

(at 0), she jumped again after the deduction of 1 to the highest value of the range (9). Just as it is the Byte type. If one subtracts 1 from a minimum value of the range, the number jumps to the highest value. So the 255 declared. But why a 5 is a "4". It's simple: 255 was added to 5 (in line 30) and the result would be 260. Since this is out of the range of bytes, so you can often deduct 256 until you are in range again. And 260-256 is "4", Consequently, one can also expect a Byte to sign limited. The same is true for the data type Word. For it -1 is the same as 65535. But be careful!

Not all operations allow this. It works in the addition, subtraction and multiplication, but not in the division. It function "=" and "<>", but not "<", "<=", ">", "> =". In type Byte -1 is defined as greater than 100, because -1 is identical to 255. Therefore, in these cases you should preferably work with "=" and "<>".

2.3.8.2 The Integer type and its similarity to Word

No such problems one has the Integer type. It can take values between -32768 and +32767. In this area, all operations work correctly and as expected. This also applies to "<", "<=", ">", ">=" (these operations are Integer type but relatively slow) and the Division. Since this model also like Word has exactly two bytes of memory, it seems likely that they are similar in some way. And in fact they are not only similar, but actually identical. Because whether an Integer variable assigns a value of -1, or a Word variables does not matter. In both cases is exactly the same in memory. How are these types differ at all? Only by the operations "<", "<=", ">", ">=", the division and the conversion to a string (with PRINT or STR$) or for Real. Otherwise, these types are totally right! Thus, for example, "I=55296: POKE I,1" possible with "I" as an Integer variable. Since the data type Word is the more important, by the way changed arbitrarily to Word in a type conflict. But such a conversion of Integer to Word or Word to Integer consumes neither time nor memory because no conversion takes place.

2.3.9 Types in operations, functions, and commands

The expected and provided by operations, functions, and types of commands you should know, so no unnecessary conversions are made, wasting the time and memory.

2.3.9.1 Types for commands

Many commands of Basic V2 expect one or more parameters. These parameters are the interpreter always of type Real. But the compiler expects a particular functional type. If the user specifies a print of another type, a conversion is made. When a particular type is expected, then the type is here easily substituted for the parameter.

2.3.9.2 Store instructions

POKE Word, Byte
SYS Word

2.3.9.3 Commands for program control

GOTO Number
GOSUB Number
ON Byte GOTO Number
ON Byte GOSUB Number
"Number" is a number of type Word and not a variable.

FOR N=A TO B STEP C
"N" must be a variable of type Byte, Word, Integer or Real (ie numeric). A, B, and C should be of the same type.

```
NEXT N
"N" must be the same variable as FOR.

IF A THEN ... Number
IF A GOTO Number
IF A THEN Number
"A" can be an expression of any type.
```

2.3.9.4 Input and Output (broadly defined)

```
PRINT   A
INPUT   V
GET V und READ V
OPEN    Byte, Byte, Byte, String
CLOSE   Byte
PRINT#  Byte, A
GET#    Byte, V
INPUT#  Byte, V
LOAD    String, Byte, Byte
SAVE    String, Byte, Byte
CMD     Byte
"A" can be an expression of any type.
"V" is a variable of any type.
```

2.3.9.5 Additional Commands

```
DIM     V (Number,...)
DATA    D, D, D
RESTORE Word
WAIT    Word, Byte, Byte
The data "D" are normal strings or numbers
```

2.3.9.6 Types for functions

Functions differ from commands in that they deliver results. These results are of a particular type. SIN, COS, TAN, ATN, LOG, EXP, RND, RND, USR each expect a real number, and also supply such (for example, "Real =SIN(Real)". SGN, INT, ABS can be applied to all numeric types. The result is a number in this type respectively.

```
Byte   = PEEK (Word)
Word = FRE (X)
Byte   = POS (X)
"X" is completely insignificant (dummy parameter)

Byte   = ASC (String)
Real   = VAL (String)
String = CHR$ (Byte)
String = STR$ (A)
"A" is any numeric expression

Byte   = LEN (String)
String = LEFT$ (String, Byte)
String = RIGHT$ (String, Byte)
String = MID$ (String, Byte, Byte)
```

In this division, Arrays fit. You expect parameter of type Word and deliver the type of the Array variable. For one-dimensional Arrays of type Byte, Boolean, Integer or Word, the

parameter type Byte is expected if the dimension of the Array is correspondingly small (then an efficient addressing mode may be used).

2.3.9.7 Types in operations

The following operators provide the same type with which they are supplied: plus (+), minus (-), AND and OR process all numeric types. Multiply (*) and Division (/) processing Word, Integer and Real. Exponentiation can only with real numbers. The negation (-), and NOT handle all types from Bytes to Real. AND, OR and NOT also handle all numeric types and additionally Boolean. There is also a group of operators that do not provide the type you would expect. These are the comparison operators. They handle all types (including String), and generally deliver the type Boolean.

2.3.9.8 Types of system variables

The C64 is familiar with the system variables TI, TI$ and ST. TI is of type Real, TI$ is of type String and ST is treated as a Byte.

2.3.10 The data type "Constant"

Pascal and Modula offer the programmer the ability to replace difficult to interpret numbers with meaningful names. Basic does not have this advantage. Here one must instead use variables if you want to write any numbers. This is particularly unfortunate because the efficiency of compilations when using constant numbers instead of variables increases significantly. The code is faster and shorter. "POKE COL,1" is compiled as three times as long and slow as "POKE 53281,1" Therefore I have enriched the Basic-Boss the data type of Constant. A constant is declared like any other variable, which is specified as the data type "CONSTANT". It may variables of the form "A", "A%" and "A$" are declared as constants. Like any other variables of a constant in the program can be assigned a value by placing an equal sign after the variable name, followed by the desired value. In this way, the compatibility is maintained to the interpreter. However, a constant may be assigned a value only once. This must also be done prior to their use (ie, in a previous program line). Therefore it would be best to assign its value to the constants in the program beginning. Then the constant is ready for use. It can be used in the following program as often and then each of its assigned value replaces.

```
Example:    10 rem@ £constant col
            20 col=53281
            30 poke col,1
```

This program is in the exact same view of the compiler comprising:

```
30 poke 53281,1
```

In the first program, the line 30 is considerably more readable. You should be in the use of constants aware that they don't have any particular type by default. When assigning a value to a constant, only the text is actually read and noted. Therefore, you can assign a constant everything possible (for example, "C =HELLO" or "C =4+1.23*5"). If this constant shows up later in the program text, then their assigned text is bracketed and used. Only then the expression is evaluated. If there were errors in the constant text, they are therefore not reported until now. It is necessary to use parentheses. Assume "C" is a constant and was determined: "C=1+2". Later, when the expression "A=C*3" appears, without parentheses the result of "A=1+2*3" would be 7. But what is expected of "A=(1+2)*3" is 9. Constants can be nested. For example, if VIC and SPR are constants, then these also as evidence: "VIC=53248: SPR=VIC+21". If SPR is used in the following program text, SPR is first replaced, VIC and the result is for "POKE SPR,255" of the replacement text "POKE((53248)+21),255".

2.4 Description of the directives

The settings and properties of the compiler can be changed only with compiler directives that are in the basic text of the source program. This increases the flexibility and the ease of use of the Basic-Boss, since you do not have to re-adjust as with some other compilers for each compiler run program-specific parameters. In the application of compiler directives, you should always be quite clear that the compiler of the Basic text (which is determined by the GOTO and GOSUB) works through line by line from front to back and does not care about the actual execution sequence under the interpreter. The directives should therefore always be at the beginning of a program. A number of directives can often be used as opposed to anywhere within the program. If this is the case, it is mentioned in the description of the directive. The following compiler directives are explained according to their function. They are written in capital letters and each of them must have a pound (£) be prefixed so that they are executable. If you do not necessarily want all the possibilities of Basic-Boss, you only need to see the commands that are designated as important.

2.4.1 Control of the data types

The following commands are very important:

(£BOOLEAN), £BYTE, £WORD, £INTEGER, £REAL, £STRING

These commands are followed by a list of variables of the form "A, B%, C, D$" with any number of elements. Instead of variables, "%", "$" or "#" can represent for all variables of the form "A%", "A$" or "A". The affected variables are declared for this type. Behind a variable, an equals sign, which in turn must follow "FAST" or an address. These commands must be at the beginning of the program. More see Section 2.3 "Data Types". The next command you should also know if you work a lot with DATA:

£DATATYPE Type

Thus, the type of the data DATA of the instruction is set. By default, all data of type String is stored in this type in the compiled program. Because of the type String can be converted by the READ command to all other types. If the data but in reality, for example, are numbers between 0 and 255 (which is very often the case), and are meant for a variable of type Byte so it would certainly make more sense to store the data already in this type. This happens after the directive "£DATATYPE byte" has been found. Then the data after the DATA command should be numbers between "0" and "255". You will then be stored directly in the type Byte, which saves a lot of memory (approximately 70% to 80%) and increases the execution speed of the compilation dramatically. You can also use any other data type instead of "Byte", which then require correspondingly more memory ("REAL" for example, requires five bytes per data value and thus saves more memory to the String type, but this time). This directive can be often used in the same program, as required by the different data. However, it should be noted that the READ command, the numeric types can not converted into one another. Therefore, the compiled program gets off with a "type mismatch error" if you want to read Byte data into a Real variable, for example. The problem can be solved by first reads the Byte data into a Byte variable and then assign the Byte variable to a Real variables, which is still faster and especially space-efficient than using normal String data.

For example:
```
10 rem@ £byte a,b,c
20 read a,b,c
30 read x$,y$,z$
40 rem@ £datatype byte
50 data 10,200,50
60 rem@ £datatype string
70 data dies,sind,strings
```

The following directives are fairly unimportant and are rarely needed.

£PRINTTYPE Type

It is set to the specified type, the minimum type of calculation in the PRINT command. By "£PRINTTYPE Real" that is, all operations in the PRINT command are carried out at least in the Real type. If "W1" and "W2" are word type and the values include "5", "2", the command "PRINT W1/W2" no longer provides the incorrect result "2" but "2.5". By default, the type is Boolean.

£BOOLEANTYPE Type

This command sets the minimum type in the calculation of IF-fixed expressions (eg, "IF A+3.5=B THEN ..."). This minimum standard type is Boolean.

£CALCTYPE Type

This type is generally the minimum specified in calculations. Normally it is set to Boolean. To establish the full compatibility to Basic, you can set it to Real (with "£CALCTYPE Real"), but what the programs unnecessarily slow down.

2.4.2 Optimization

The next two commands you should know if you want to write fast programs:

£FASTFOR and £SLOWFOR

The FOR-NEXT loop in the compiled program works very cumbersome and slow because it is supposed to work exactly as in Basic. Therefore, the Basic-Boss offers an alternative FOR-NEXT loop, which is much faster, but has a few limitations. These are the following: FOR and NEXT must be in the correct sequence in the program text and on each FOR NEXT exactly one must follow (similar to Pascal, Modula or C). The following constructions can the Basic-Boss problem about it when it is to produce fast grinding:

```
10 goto 30
20 next i: end
30 for i=1 to 100
40 print i: goto 20
```

or something like this:

```
10 for r=1 to 100
20 print i;
30 if i<50 then next i
40 print "abc"
50 next i
```

Another limitation is the type of the counter variable. It may only be of type Byte or Word. Otherwise, the Basic-Boss used the slow loop. Also in the STEP value, all is not allowed, if it is specified, it must be stated directly, that is, as a number and not as a variable. He may, however, be both positive and negative. Moreover, the value of End variable must not change. In "FOR I = A TO E", the value in "E" must always remain the same during the execution of the loop when the loop in the compiled code the same shall make such by the interpreter (unless "E" is an expression). Despite all these limitations, the FASTFOR command on the vast majority of loops can be applied. Normally, the Basic-Boss slow loops (because of compatibility) produced. But if he finds the directive "£FASTFOR" he produced from then on fast loops. When he comes back to "£SLOWFOR" he produced slow but fully compatible loops again. If you want to make only a single loop faster, then so you can write on this loop "£FASTFOR" and beyond "£SLOWFOR" One should note that in "£SLOWFOR" all loops are completed with NEXT, since the Basic-Boss otherwise returns an error. One should always

specify the loop variable after NEXT, because the compiler can detect errors better. Nests are of course also allows for fast grinding. It is at FASTFOR loops as opposed to slow loops also unproblematic to jump out of the loop with GOTO. The next four commands are for the optimizer to you and not essential.

£LONGIF and £SHORTIF

There are two ways to construct an IF branch for the compiler. The method allows an unlimited number of commands after the THEN, the other not. This saves the second process, however, for each three-byte memory IF instruction and the code is a bit faster. By default, the compiler applies to the LONGIF process, which allows an arbitrarily large IF branch. With "£SHORTIF" one switches to the cost-saving version, but sufficient in most cases. Therefore I would recommend applying to program with "£SHORTIF". If the Basic-Boss but then again strike at an IF line (which indicates an error), then you can switch on this IF line with "£LONGIF" on the longer version, and behind it with "£SHORTIF" back to the shorter. The following two commands can be important when testing:

£SLOWARRAY and £FASTARRAY

If a field (Array), for example, with "DIM A(20)" is dimensioned, it allows the compiler for simple and fast array that the Basic program accesses to non-existent array elements. The statement "X=30: A(X)=1000" will be executed without difficulty, Whereby other variables or memory contents will be destroyed. However, to compensate for such arrays are extremely fast. If you still want to play it safe, so you can do this with " £SLOWARRAY" what the program then slowed down. "£FASTARRAY" on the fast array management again. Both commands can repeatedly appear anywhere in the program, so you can dose the security target.

2.4.3 Setting the start addresses

These commands should only apply if you have some basic knowledge regarding the memory layout of the C64. Otherwise, you can ignore them. Usually, the compiled program is stored in a single file whose name matches the Basic-Boss receives from the name of the source program by preceding it with a "+". The compilation is compiled to the normal Basic start (2049) inclusive SYS-line so that you can start it with "RUN". But there is another way. You can lay the compiled program to any location in memory. You can even split it into its components, and this creates any memory addresses and writes under any name on disk. So it is even possible to keep multiple compilations simultaneously in memory that call each other when they are placed at different addresses. To all this may come an ordinary Basic program that may be using the compilations as subroutines and controls. The memory layout of compilations usually looks like this: It consists of three parts. These are: program part, the routine part and the data part. The program portion represents the actual machine program that was created from the Basic program. This part of the program requires for its working routines of the Basic-Boss individually obtained from his library and writes behind the program part, the compiler conforms to the address range (this process is called "left" [Linken]). Then follows a data area, are in the information about array variables and a table for the READ command. Finally, the variables follow memory, the memory area for help and eventually the string memory. Although this sequence appears to be the most reasonable, but not the reality. Or more precisely, it is only correct in pass 1. In pass 2, the compiler first routines and data area and then install the program area. This requires extensive calculations, but it has a significant advantage: The routine area is in the compiled program at the front. This ensures in the normal case, that the routines are in the lower 40-Kbytes range. And that's where they need to be, because they must necessarily be in a storage area that is not superimposed from ROM or from others.

The position of the regions can be changed using the following commands. It should be noted that there is no overlap in each area. If the position of a code area is changed, then it is written to a separate file. An old file of the same name is deleted by default (with the SCRATCH command).

£ROUTSTART address

Defines the beginning of the routine area. The routines are not to be placed under the ROM or the I/O area. The file name of this part is located by default in "R+" and the name of the source file. The SYS-line is part of this range. Therefore, the command also turns off the SYS line, if it was not explicitly enabled with "£SYSON". Then the routine area can be jumped directly to a SYS command when the compiled program is to be started. By default, the routine range is 2049, ie at the beginning of the Basic memory.

£DATASTART address

Specifies the starting address for the data area. The file name of the data area is derived from "D+" and the name of the source file.

£PROGSTART Address

Specifies the starting address of the actual program. It can be laid anywhere, for example, to $D000, if the compiled program in the "£ALLRAM" mode runs. The file name is derived from "P+" and the name of the source file.

£VARSTART address

Determines the address of the variable memory.

£HELPSTART address

Determines the beginning of the memory help.

£HEAPSTART address
and
£HEAPEND address

Determine the location of the string memory. The address for "£HEAPEND" are no longer used at the first memory location. If 0 is specified for "£HEAPEND", no string memory is reserved at all, which is useful if you want to execute any string operations. Variables, auxiliary memory and string as the program area can be laid anywhere. Usually, all areas are concatenated directly. If you have created your own Basic-Boss version using "£SETPREFERENCES" that does not concatenates the areas, so you can force "0" as the start address of an area with indication of this".

2.4.4 The files

The file names of the output files, you can specify your own, if you do not like the default name.

£ROUTFILE device, "filename"

So the name of the file for the routine part sets, usually takes the name also from the data part and the program part. The device is useful as either "8" or "9", you will usually omit it (£ROUTFILE "filename"). Behind the device could also specify the secondary address, but which should be useful in very few cases (only for particularly troublesome RAM disks or similar).

£PROGFILE device, "filename"
and
£DATAFILE device, "filename"

Function as the program area and the data area. The name of a file can be safely followed At sign (@) are preceded by a colon. The Basic-Boss recognizes this and then deletes the old file before opening the new with the SCRATCH command of the floppy. This is necessary because the overwriting with the At sign (@) because of an error in the floppy does not work

normally and should generally be avoided, otherwise files may be destroyed. If the file name is preceded by no At sign (@), then an old file of the same name is not deleted and it may occur a floppy error. There is another command file:

£SOURCEFILE device secondary address, "File Name"

This command seems plenty of sense. It specifies name, unit and secondary address of the source file after sameness is already open (yes it is read from it). Therefore it only makes sense against a "£SETPREFERENCES" command. Secondary address, and / or device may be omitted as usual.

2.4.5 Logging of compiler activity

Also, the following directive is not of overriding importance, but should still be in every compiler or assembler.

£PROTOCOL device, secondary address, "File Name"

The error messages and other data (such as the memory addresses) usually appear only on the screen. If you want to guide them to the printer, the directive "£PROTOCOL" enough. If the printer has a different device address as "4" (for example the plotter 1520), then you should specify this with "£PROTOCOL device". To the device, you can still attach the secondary address and possibly a file name (for example in Görlitz-Interface this controls the line spacing). "£PROTOCOL 8,"ERROR", S, W" sends the error messages into the "ERROR" file on disk (again, you can safely use the At sign). This file is a simple ASCII file and can, for example, with the program "ERRORREAD" (on the program disk) be read again. If "£PROTOCOL" was used the compiler will not produce errors anymore. With "£PROTOCOL 0" You produce this effect without sending any output.

2.4.6 Memory Management

The Basic-Boss allows you to use all the memory of the C64 so "PEEK" and "POKE" will be slower. If that bothers you, then you just set the directive "£ALLRAM" at the beginning of the program and the matter is settled. But if you want optimum speed, then you should deal with the commands "£RAM" and "£ROM".

£ALLRAM

Must be at the beginning of the program and instructs the Basic-Boss to use all the memory for the program. In addition, the string end memory is automatically set to the end of memory, if not a "£HEAPEND" command is in the program. In ALLRAM mode, the full 64 Kbyte RAM of the C64 are always switched on (usually only the bottom 40 Kbytes of on and a small area starting at address 49152). However, then it must first be switched on all PEEK, POKE and SYS commands on the ROM, so that in each case they provide the same result as the interpreter. This slows down of the program is insignificant. When one of these commands, but anyway want to address only the RAM and not the ROM, the I/O memory area or the color, then you can prevent the switch with £RAM. Each POKE, PEEK, or SYS command then refers to the RAM memory. If you want to access the ROM or the I/O blocks again, you have to turn on the switch before with £ROM again. If you forget this command, for example, the screen color is not set, but rather the string memory disturbed and the garbage collection runs crazy. These two commands are useful only for "£ALLRAM" "£RAM" and "£ROM" can be used as many times in the program. A supporting program ("RAMROM") is on the program disk. Also in the program "RAMLOAD" makes use of "£ALLRAM" and "£RAM". A more efficient way to access the ROM in ALLRAM mode provide the commands "<-RAM", "<-ROM" and "<-IOROM", but should only use the professional (see 2.5 "New commands and functions").

£BASICRAM

Is the counterpart to "£ALLRAM" and causes only the normal Basic memory for programs and variables used. If the string end ("£HEAPEND") has not yet been set, it is reduced to 40,960 (end of basic memory). This is the normal state.

In ALLRAM mode the compiled program may extend beyond the Basic area, and may be longer than 154 blocks. Then there is also the RAM under the ROM. A length greater than 201-202 block is therefore not possible, because the ark of the C64-routine overwrites the I/O area, which can be visually identified by a drastic change in the screen display. In this section, a program can only be loaded with the compiled code of "RAMLOAD". This RAMLOAD may even not be overwritten.

2.4.7 The SYS-line

£SYSON and £SYSOFF

The SYS-line makes it possible to load the compiled program as a normal basic program and launch it. It can be prevented by "£SYSOFF". However, this is usually not necessary because the "£ROUTSTART" the SYS command line switches off automatically, so the program with "SYS start address" can be started from the direct mode or from another program. With "£SYSON" can force the SYS line. The following commands have only purely cosmetic significance:

£SYSNUMBER number

Determines the line number of the SYS line.

£SYSTEXT "Text"

Sets the text in the SYS line.

2.4.8 Other directives

£LINEON and £LINEOFF

Switch the row updating on and off. Normally it is disabled because it consumes memory and slows down the program. If it is on, the errors occurring while the program is running with the correct line number are displayed, otherwise not. The following commands are now for the occasional user of lesser importance:

£LONGNAME and £SHORTNAME

In Basic arbitrarily long names for variables are allowed. Nevertheless, the interpreter distinguishes only the first two places. "KUCKUCK" and "KUHFLADEN" for the interpreter are exactly the same. The compiler is designed for however arbitrarily long variables. For compatibility reasons, but he distinguishes only two places. With "£LONGNAME" can remedy this deficiency, if one takes into account that the program no longer runs properly under the interpreter. "£SHORTNAME" switches back to double-digit name.

When using long names should make sure that no Basic command is included in the name, as this inevitably leads to an error.

£IGNORE and £USE

"£IGNORE" tells the Compiler to ignore everything below. Only when he meets "£USE" he examines the Basic program. This function is useful if you want to hide some parts of the Basic program of the compiler. This can be very useful when the running ability to be retained under the interpreter and you still want to take advantage of special compiler features.

Example: 10 print "for compiler and interpreter"

```
            20 rem@ £ignore
            30 print "only for interpreter"
            40 goto 70
            50 rem@ £use
            60 print "only for compiler"
            70 print "again for both"
```

£SETPREFERENCES

If you do not want to put new in each program by compiler directive his favorite settings (for example, start-up, default data type and so on), then you can write a small program that sets the desired settings, and then the committed "£SETPREFERENCES" command. The current settings of the Basic-Boss are then copied to the default settings and the compilation process is aborted. If you now with the Basic-Boss << -> restarts, you can enter an asterisk ("*") followed by a name under which your individual Basic-Boss is then stored (not the original disk!). In the future, you can then use this Basic-Boss version instead of the standard version and will not have to re-set the most commonly used settings. A sample program can be found under the name "PREFERENCES" on the disk.

£CODE value, value, ...

This command is intended for expert on the machine language. The command word followed by any number of comma separated values. This can be Byte values. It can also be password-values if the numerical value is "w". Strings are also possible if one includes in this " " ". These Bytes or Words or character strings are inserted at the current position in the compiled program. In this way, one can generate all of the machine language instructions of the 6510. It is not need to take care of the register values (A, X, Y) as the Basic-Boss does not require this. If one uses the "£CODE" command careless, then the compiled program is then not run. A few examples: "£CODE 234,234" inserts two NOP instructions, which causes a small time delay. "£CODE 0" sets a breakpoint. "£CODE 96" is identical to "RETURN". "£CODE 108,w40962" performs "JMP($A002)" and "£CODE 2" creates a secure system crash.

£INITOFF and £INITON

All variables are known to be set at the beginning of a program to "0" (FAST and address variable excluded). This can be prevented by "£INITOFF". This command must be always at the beginning of a program. If he is found before the code generation, then no variables are initialized at the start of compilation and have a random value. When this command is used, you should refrain from use strings, as they absolutely need an initialization. "£INITON" switches the initialization again.

2.5 New Commands and Functions

The Basic-Boss sees some new commands and functions. The commands initiated with a "<-" and the functions with the At sign (@). Since the Basic interpreter does not handle both, you can "£IGNORE" and "£USE" here put to good use and provide for interpreters and compilers, different instruction sequences, so that the program will continue to be tested properly under the interpreter.

2.5.1 Input and Output

<-LOAD "FILENAME", DEVICE, ADDRESS

This command loads a file to the specified address from the specified device. If the address is missing, then the file is loaded to the default address. If one omits also the device, the device address "8 is" accepted. The program will be processed after this command in difference to normal LOAD command regularly at the next instruction.

Example: 10 print "now loading"

```
20 <-load "sprite",8,832
30 print "Load finished"
```

2.5.2 Efficient input and output

The Commodore Basic allows the programmer input commands GET# and INPUT# and the output command PRINT#. This is sufficient for most applications. However, these commands also have some disadvantages: numbers are converted to strings only when PRINT# and then saved as a text, which consumes a lot of time and space. Similarly, the input is slowed INPUT#. For each command execution is switched between input or output channels. This has a very unfavorable especially with GET# on speed. It must be frequently worked with the relatively slow strings, although this would not be necessary.

@BYTE, @INTEGER, @REAL, @WORD

<-BYTE, <-INTEGER, <-REAL, <-WORD

The solution offers a few new commands and functions Input and Output. The commands to issue consist of the usual "<-" and the name of a numeric data type. This is then followed by one or more comma separated expressions (mostly single variables or numbers). The result of this expression is then converted to the specified type (if necessary) and written to the currently active output channel. "<-BYTE 3" are, for example, a single Byte that has the value "3". "<-REAL 1.2345" is made up of five bytes that match the memory size of 1.2345. Just also work "<-WORD X" and "<-INTEGER X", the write exactly two bytes.

The input is in contrast to issue commands not implemented, but via functions, because they can be applied more flexibly and more efficient. These functions always begin with a At sign (@), which similar to the commands follows a numeric data type. No parameters are required, so you can treat the functions like variables. On "A=@BYTE" a Byte is fetched from the input channel and assigned to variable A. When "PRINT @REAL" five bytes are read and output as a floating point number on the screen. "POKE 1,@BYTE" brings a Byte and writes it to address 1. These new commands and functions are in the form but not particularly useful, since they only write to the standard output device screen and keyboard can read from the standard input device. This defect helps following commands:

<-OUT, <-IN, <-RESET

"<-OUT n" directs the channel output to the file n (as well as "CMD n").
"<-IN n" sets up the input channel to the file n.
"<-RESET n" switches the channels back to the default screen and keyboard devices.

If, for example, the 1000 elements of the real array "R(" wants to write to the file "TEST,S" this can look like this:

```
1000 open 1,8,2,"test,s,w"
1010 <-out 1
1020 for i=0 to 999: <-real r(i): next i
1050 <-reset
1060 close 1
```

When a read or write operation is enclosed, the "<-RESET" command must always be executed, so you can control screen or keyboard. Because the commands "GET", "INPUT" and "PRINT" each refer to the current input / output channel. Nevertheless, one should rarely possible reset due to time issues. The operating system does not allow for that input and output channel to be diverted. If the input channel has been diverted, so a "<-RESET" must always be made first before the output channel can be deflected (or vice versa). You can also write data of different types in the same file, if you meticulously paying attention to the data to be read in exactly the same order as they were written. The program disk are two examples on this topic called "BOSSDIR" and "RAMLOAD".

2.5.3 Memory Management

<-RAM, <-ROM and <-IOROM

These three commands are intended only for specialists. It makes the most sense to use them along with "£ALLRAM". "<-ROM" on the ROM and the input / output area (video Processor, SID, Color RAM and so on), then that may be it accessed very quickly (with poke or peek in connection with the compiler directive "<-RAM" the one precedes "£"). It can no longer be accessed in the RAM from $A000 to $BFFF and $ D000 to $FFFF. Therefore, neither program variables may still be in this range. "<-ROM" only switches the input / output range and the operating system, but not the Basic interpreter. Can no longer be accessed in the RAM from $D000 to $FFFF. "<-RAM" switched back on all of the 64 Kbytes. By "<-ROM" or "<-IOROM" only simple operations with Word or Byte types should be performed. Each complex operation calls a subroutine that returns to the RAM mode in general. Instead of using these commands, you can also change the way safely, even with "POKE 1,X" perform (but also only in ALLRAM mode).

<-SEI and <CLI

These commands are more suitable for more professional programmers. They are identical to the same machine language instructions. With "<-SEI" you can lock the system interrupt. This is useful when programming particularly time-critical applications that can not tolerate brief interruptions. However, the keyboard no longer prompts for "<-SEI" and the time is not counted. Thus, one should turn to interrupt with "<-CLI" as soon as possible. You can also be used sensibly using these commands "<-RAM ","<-ROM" and "<-IOROM" if the program is not running in "£ALLRAM" mode. You then should avoid complex operations. Otherwise called subroutines may switch back to the ROM mode. You should run these commands very carefully and do not perform Input and output operations by "<-SEI".

2.5.4 Processing of commands from Basic Extensions

The Commodore Basic V2 is well known, is quite spartan and certainly not famous for his command range. Meanwhile, you will quickly realize when trying to program graphics or sounds. So it is not surprising that there are a variety of extensions and additional routines that address this deficiency. From the perspective of the Basic-Boss, there are basically three different types of such extensions: The simplest are called with "SYS address". Possible parameters are previously placed by poke into certain memory cells (eg POKE 900,A: SYS 49152") Its parameters a little more user friendly is the second type, which is also called by SYS, but is appended to the SYS command expected (eg "SYS 49152, A, B$, C"). The last type is the most convenient because it works with completely new commands (eg "LINE X1, Y1, X2, Y2"). The simple extensions make the compiler no difficulties, since only completely normal Basic commands are used. Just as willing to process the USR function. In both cases, no special precautions are necessary provided that the extension does not use major parts of memory (in which case you may need to work with £HEAPEND).

£OTHERON and £OTHEROFF

It becomes difficult for the compiler with additional SYS-parameters or even whole new commands. Then special precautions must be taken, including one causes the Boss "£OTHERON" at the beginning of the program. If the compiler is now an unknown command (more tokens or token sequence), then it assumes that it is the command of an extension. The Basic-Boss evaluates the unknown command including parameters and delimiters and writes all together in a special format in the compiled program. The appended to the command parameters from each other by commas or any other special characters (such Example, be separated with a basic command) (eg "LINE X1, Y1 TO X2, Y2"). Even more separators or none are accepted. But the order and the type of delimiters to be expected from the expansion, as well as the command itself, since otherwise the compiled program a "SYNTAX ERROR" occurs. Because the compiled program executes the command does not itself but passes it to the Basic interpreter. If an extension is in memory, which is properly integrated into the interpreter, then they will process the command. The extension can be passed only Strings and Real numbers, because the

interpreter knows only these types. Other data types are therefore converted to Real. Above all, it should be noted that the extension should only have read access to the variables. A machine routine for sorting strings can, for example, are already not used solely for the administration of the string completely new Basic-Boss. "£OTHERON" by the way also works in the ALLRAM mode. However, then the transferred string can not be longer than 80 characters and you should note that the extension also their storage required, the one you should secure as with "£HEAPEND". "£OTHEROFF" disables the management of SYS extension parameters and commands. This is the normal state. For those interested a word to the principle: the transfer of SYS-parameters is actually not readily possible because the variable structure of the Basic-Boss that of the interpreter is different too. Fortunately, the interpreter provides a vector for fetching an arithmetic element. This vector is directed from the compiled program on your own routine in "£OTHERON". Prepares you for the Basic-Boss variables and fed it to the interpreter, who in turn passes it on to the machine program the extension. The separator character reads the extension itself. For new orders still come into play another vector ($308/$309), over which the compiled program starts expanding.

£OTHER

Some extensions to work despite "£OTHERON" with the compiled together and not only produce "SYNTAX ERROR". This applies to a certain class of extensions that it is often seen that their commands are preceded by a special character, for example Left Arrow, exclamation mark or brackets or similar (there are no real token used). If this is not the case, then you should invite her Basic program even without the expansion in memory. If the extension commands are now still perfectly good to read, then your extension belongs to this group. Their commands can not handle without help, because he does not know where the command to stop and start the parameters of the Basic-Boss. If the instruction for example "!COLA" is, then either a single command called "!COLA" or "!COLA" mean, with A as a variable. The "£OTHER" directive creates the remedy. You should follow a list of all the commands used in the progam, which are separated by commas, for example "£OTHER !COL,!PLOT,!HIRES". If a program line is not enough, also several other directives can be used. However, for safety's sake you should not be in REM lines, but are kept by the interpreter by GOTO (for example 10 GOTO11: £OTHER !COL"). Then the extension commands may also contain normal basic commands (for example "!TEXTON" contains "ON "). "£OTHER" must stand before the first extension command (otherwise the "7/CODE MOVED" error is possible). In the modern extensions of the OTHER command is not required (for example HiRes-Master from "The Best of Grafik 3").

£BOSSOFF und £BOSSON

In rare cases, it may happen that an extension command unit with a Basic-Boss command in conflict. If an extension command for example "<-LOAD", the Basic-Boss will incorrectly assume that this is meant for him. So you can switch off with "£BOSSOFF" all directives and orders of the Basic-Boss. With the exception of "£BOSSON" because as you switch the commands again. Both directives can arbitrarily are often used in the program. Which extensions work?

My experience, comprehensive Basic extensions such as Simons' Basic and G-Basic does not work together with the Basic-Boss because they interfere too deeply into the internal structure of the Basic. The typical graphic enhancements such as Grafik 2000 (The Best of Grafik 2), or quite fast HiRes-Masters (The Best of Grafik 3) working against it very well with the compiled program. In HiRes-Master is to make sure that you bring the strings "£HEAPEND $8000" in security. Both extensions require no Other command (only "£OTHERON"). Even the Scroll-Machine (The Best of Grafik 2) cooperates with the Basic-Boss. However, here the Other directive is needed. Other extensions use partially the Basic start up to make room for sprites or graphics. Again, this is not a major problem. One has only with the Basic-Boss "£ROUTSTART address"

tell where the start is basic (as compilations, that machine programs as opposed to Basic programs are not relative). With loaded extension can be the starting address with "PRINT PEEK(43)+256*PEEK (44)" learn (the end address for £HEAPEND with "PRINT PEEK(55)+256*PEEK(56)"). Otherwise is still important to ensure that the extension fetches the values of the variables and not writes herself into it. When it is used third-party extensions or ... it should not be used fast variables because the memory locations $FB to

$FE remain without changes. When using interrupts the extension (for example raster interrupts to split screen), then you should never use "£ALLRAM". If in doubt, however, helps only try. The compilation is like a normal BASIC program loaded and started. Of course, the extension must be in memory. A demonstration program with a small sample extension is in addition to source code on the disk under the name "OTHER".

2.6 Changes in Commodore Basic V2

Due to the different working principle of interpreters and compilers result in some of the normal commands differences or new features. I have also improved a few commands. All that is listed below.

2.6.1 Improvements

At a few points I have made improvements or corrections. These expand the possibilities, but do not cause the compiler incompatibilities that arise. However, a basic program does not work properly under the partial interpreter when these opportunities.

Hexadecimal numbers

It is now allowed to specify numbers in hexadecimal notation. A hexadecimal number starts with "$", then may follow any number of digits (0-9 and A-F). But this way only whole numbers can be specified.

A=ASC(A$)

The ASC function was slightly modified. If A is empty, then returns the ASC (Byte) 0, and there is no "illegal quantity error". This makes sense, because at "GET A$" is a null character becomes an empty string.

A=FRE(X)

This function returned the interpreter the size of the free space. The result was occasionally negative (when the area was greater than 32 Kbytes). Well it basically provides the size of the remaining free memory string and positive impact.

RESTORE

Can now have a line number as a parameter. If the next READ command should refer to the example on the line 150 so you can now write "RESTORE 150". Thus the data does not have to necessarily read from front to back and no longer need so embarrassing to pay close attention to the order.

READ and DATA

Does not change the syntax. Nevertheless, these commands have become a good deal more powerful. The compiler directive "£DATATYPE Type" allows already store the data for DATA in its determination type, which saves memory and processing speed significantly. You can find further under 2.4.1 "Control of Data Types" in " £DATATYPE".

FOR and NEXT

There is a second version of the FOR-NEXT loop that can be activated by "£FASTFOR". For details, see 2.4.2 "Optimization" in "FASTFOR".

2.6.2 Commands with loss of meaning

Some basic commands are primarily intended for the direct mode, which is why certain difficulties in their implementation occur in the compiled program.

LIST, NEW, STOP

All three commands are completely identical to the Basic-Boss with END. For the LIST command in the compiled code makes little sense, since it would best show the SYS line and then exit the program anyway. NEW is pretty meaningless in the program and a proper implementation of STOP does not seem sensible to me.

CONT

is actually completely useless in the program. In my experience, it seems like an endless loop Basic program. Therefore also the Basic-Boss the occurrence of such CONT produced.

LOAD, SAVE, VERIFY

They work the same as under the interpreter. LOAD loads data into memory and jumps to the beginning of the program. SAVE saves the current program in memory from Basic and VERIFY compares it with another program to disk (and optionally generates a "verify error"). SAVE and VERIFY were implemented only for the sake of completeness.

2.6.3 Changes

Here are the commands that are not quite as responsive in the compiled program, as under the interpreter.

INPUT

The concept of the INPUT command is changed a little thing: When "INPUT A,B,C" how else can show up a "redo from start" notice, if the input is nonsensical. When interpreter, the user must then enter all three values again. When Basic-Boss he has to enter the values from the parameters in which the error occurred. The full compatibility seemed rather pointless.

IF A$ THEN

The condition is fulfilled in the compiled code, if "A$" is not empty. The interpreter responds here apparently whim (now and then with a "formula too complex error") which suggests to me that this is wrong in lnterpreter. The once published in the magazine 64'er program "TURBODIR" not as a compiled program works because it takes advantage of the operating system error. It is checked for the length of 0 where the wrong variable. Since the interpreter does not check the specified variable, but the other, it works under the interpreter anyway.

2.6.4 Problem cases

GOSUB ... RETURN and FOR ... NEXT

They work normally as expected, but crashes in a given situation "RETURN" from:

```
10 gosub 20: end
20 for i=1 to 10
30 goto 50
40 next i
50 return
```

This sequence works the interpreter properly. The compilation against crashes. The reason for this lies in the fact that in the machine language instructions "JSR X" and "RTS" the Basic-Boss GOSUB and RETURN translated, which also appears as the only sensible way.

However, these commands occupy the stack. However, since FOR sets its parameters on the stack, RETURN processes the values that were originally designed for NEXT and jumps somewhere. This could be avoided by a special identifier of the stack content, similar to the interpreter makes. That would drastically reduce the speed. Such a stack management is still in the planning stage (switchable). Incidentally, there is no "return without gosub error" as a superfluous "RTS" jumps back into the basic out of the machine program. Again, this would not be the case with an identifier.

DIM

When using the DIM command, there are only few restrictions. However, one must be quite clear that the DIM command is actually a compiler directive, which is why he is already evaluated during compilation. Therefore it is not allowed to use variables in sizing fields (for example, "DIM A(B)"). In addition, a variable before its use should be dimensioned with a DIM (even if it should contain only eleven elements), since then some more efficient code can be generated (X-indexed addressing mode). Directives from the nature of DIM results also prohibit the following construction, since the compiler is not based on the actual order of execution, but the programs by going from front to back:

```
10 dim a(10)
20 goto 40
30 dim a(20)
40 :
```

DEF FN

Since the compiler as opposed to the interpreter does not evaluate the expressions during program execution, problems also arise here. A function must be defined prior to their use (for example, in a line with a smaller number). A FN function must not be defined more than once in addition, how is that possible in Basic. Nesting of FN expression is allowed, as a mixture with variables of type "Constant". One should nevertheless very spare with DEF FN, since it can greatly extend the compiled program, for the appearance of a FN function every time it is re-evaluated. This is in the nature of FN reasons: It is a macro and not a subroutine.

LOAD

Under the Basic interpreter is a program load additional Basic programs. The program then has reloaded the memory of his predecessor, and can access its variables. This only works if the program is reloaded smaller than its predecessor and the variables are no string variables, in the form " A$ =" ... " " were set. A Basic-Boss compiled program can also reload any other program. But the successor can not access the variables of its predecessor. In a pinch, you can pass values to POKE, PEEK or address variable (recommended is the range 828-1023).

PRINT A, IF A THEN and STR$(A)

In all three commands can occur in rare cases, problems with the data types of the Basic-Boss. Because these three commands the lowest type is the minimum type Boolean, ie at all. If "A" is an expression, that expression is calculated in its type. The command sequence "W1=5: W2=2: PRINT W1/W2" will not, for example "2.5" but output "2", if "W1" and "W2" are Word variables. Meanwhile, you should be aware of when using these commands. You can change this with the directives "PRINTTYPE" and "BOOLEANTYPE".

2.7 Size and Speed

The length of the program varies approximately between 80% and 170% of the length of the original program. One can safely reach factors of 10% or 1000% if it intends to. Normally, however, this will never be the case. If programmed effectively varies the factor is between 80% and 120%. Add to this the hyperlinked routines (String administration and all that is

normally in ROM). The variable range is shorter. A variable declared as a Byte needs only one byte. The interpreter is required, in principle seven bytes per variable. The speed factor varies between three and 1000. When using Word and Byte he is about 50 to 300. In Real, it's between two and eight and in String it's between five and ten. Order of magnitude corresponds to the speed of some time ago in the 64er magazine published As-Compilers, the compiled code of the Basic-Boss but it's more efficient.

2.8 Efficient Programs

This describes how to obtain short and quick compilations. This makes the Basic-Boss although fairly simple, but you must also do its part to himself. If you want to get short and quick compilations, you should observe the following basic rules:

2.8.1 Data Types

2.8.1.1 The right type

Unless it is somehow possible, the Byte data type must be used, because it is the actual data type of the C64 and therefore the fastest and shortest. The range of values from 0 to 255 is sufficient for many applications (for example, string processing). If a larger range is desirable, one should use word variables. They are half as fast and consume twice as much space in the program and variables in memory. If possible, you should also use FAST variables (especially for addresses in PEEK and POKE addresses). If you really want to do any operations with signed numbers, because you want to use ">","<",">=" and "<=" but you can't do without the operators, then you should use the Integer type of the Real type in each case, as far as possible, then the Integer is as fast as Word in most cases, except for ">","<",">=" and "<=". If these operators are not required, you can expect the numbers are signed, but where you have to watch out for conversions and the division even with Byte and Word. Only when it is unavoidable, you should use Real variables or Functions. Similarly, one should avoid Strings as much as possible. However, operations on Strings or Real numbers require an extremely economical due to parameter passing principle less program memory than most of the Word and Integer operations (for example, addition: 17 to 19 bytes in Word, 9 bytes in Real).

2.8.1.2 Transformations

It is extremely important to avoid unnecessary conversions. You should always use the types that are expected for commands or operations. Which they are, can be read under "types for functions and commands" in 2.3 "Data Types". The command "POKE A,B" for example, is horribly inefficient when "A" and "B" are of type Real.

Then, an operation (POKE) made three operations (A <-Word, B<-byte, POKE) and the speed is greatly reduced. It is similar with the other commands. There are also a few transformations that may be used arbitrarily because they require neither space nor time. These are: Boolean<-->Byte, Word<-->Integer, Integer<-->Byte and Word<-->Byte. Very little space and time require the following transformations: Byte<-->Word, Byte<--> Integer. Especially conversions to and from Real to be avoided. One should decide the data type for a variable also based on their use. For example, if the variable "A" takes only the values 0 and 1, you should not define them as a Byte anyway, but as Real, if it is to be charged exclusively with Real variables.

2.8.2 Constants, operations, commands, arrays

Constant

The constant use of numbers instead of variables accelerated processing at Byte, Word and Integer and saves space. Especially with addresses for PEEK and POKE should use constants

(or FAST variables). But if you want to use non-meaningful numbers, you could use the data type "Constant".

Operations

If you are using Word, Integer, and Byte, it's best to only use simple operations (addition, subtraction, assignment, comparisons, AND, OR). Operations such as multiplication, division or exponentiation should be avoided, but where multiplication and division for special routines for Word and Integer are used, which are still relatively fast. Essential to avoid the following functions: SIN, COS, TAN, ATN, EXP, LOG, SQR, RND, and especially potentiation. How this can often be realized is described later in 2.8.4 "Tables".

Commands

The commands were largely accelerated. This applies in particular to:

POKE, PEEK, GOTO, GOSUB, RETURN, FOR ... NEXT (FASTFOR), LET, ON, WAIT.

Even the PRINT command in Word, Integer, and Byte much faster, as special output routines exist for these types.

Arrays

Avoid multi-dimensional arrays, because they are very slow because of calculating multiplications are necessary and precisely "(dimensions-1) piece". Also, avoid arrays of type Real and String and use "£SLOWARRAY" only for testing. For one-dimensional arrays, you need not to worry regarding the speed at the data types Integer, Byte or Word, because the access is very fast on these arrays due to special algorithms. Sizing your fields always at the very beginning of the program, even if you need no more than eleven members (DIM A(10)). Because to be able to apply a particularly efficient addressing the processor 6510, the Basic-Boss must know from the beginning how big an array is. This only applies for one-dimensional arrays of type Word, Integer or Byte that has less than 128 or 256 elements. If you are using multi-dimensional arrays, you should use Constants for the rear dimensions, as far as possible. You should create your arrays so as that you use instead of "A(10,I)" the form "A(I,10)". This will save you a lot of time and multiplication, because the Basic-Boss partially self calculates the array and the compiled program must calculate only the remainder. On "A (I,10,15)" as two multiplications can be saved.

2.8.3 Avoidance of unnecessary command execution

You should write your programs so that only the commands are executed that are necessary. In particular, one should avoid the unnecessary processing of Real numbers or Strings. The following program can significantly speed for example:

```
10 get a$
20 if a$="a" then ...
30 if a$="b" then ...
40 ...
50 rem here you continue
```

by changing the following line:

```
10 get a$:if a$="" then 50
```

Once again much faster it goes like this:

```
5 if peek(198)=0 then 50
10 get a$
```

Here, it is checked if the keyboard buffer is empty. If so, no operation string is performed at all. This is the optimal solution.

2.8.4 Tables

You can greatly accelerate its programs with the help of tables. For example, consider again the following routine to set a point on the screen. "X" (0 to 39) and "Y" (0 to 24) contain the position and the routine is called with "GOSUB 1000". All variables are of type Word (because they are charged with Word).

```
1000 poke 1024+40*y+x,160: return
```

If this little routine is called frequently, then the program is quite slow, because multiplication has been used. The additions are comparably small. But it is also better. The multiplication you can save if you perform all possible multiplications before, and then used only the results. Here's how:

```
10 dim sc(24)
20 gosub 2000
30 ...
1000 poke sc(y)+x,160: return
2000 for y=0 to 24: sc(y)=1024+y*40: next y: return
```

Now the multiplication results are read at the start of the program in an array, so it can be accessed directly in the program. The same can be done for instance also for sine values or with the powers of two are often needed because sine calculation and potentiation are even slower.

2.8.5 Shortening of calculations

Explore the following program line:

```
10 a=10+20
```

When you start the interpreter with "RUN" as a result of this unnecessary addition. Because instead of "10+20" it would be more sensible to simply write "30". If you compile the line and examined the compiler, you will find that this does not perform addition. The Basic-Boss has therefore recognized that here he himself can simplify a little and already during compilation "10" and "20" added together, so the compiled code for "A=10+20" or "A=30" is exactly the same. This instant calculations, the compiler with the basic arithmetic operations "+", "-", "*" and "/" to perform. It can also potentiate or negate (for example -(1+2)). All these calculations are always performed in Real type. Even the expression "1+2+A" can simplify the compiler. But on "A+1+2" he has trouble. Therefore one should "A+(1+2)" write if you want to have efficient code.

2.8.6 £FASTFOR, £DATATYPE, £SHORTIF

As far as possible, FASTFOR should be used (see 2.4.2 "optimization possibilities" and "£FASTFOR"). When using READ and DATA be sure to specify the type of data with "£DATATYPE Type" (for more on "£DATATYPE"). In addition, "£SHORTIF" is recommended.

2.9 Adaptation of existing programs

Probably you have basic programs that you wrote at a time when you knew nothing of Basic-Boss. If you want to adapt such a program to the Basic-Boss, then gives a number of characteristic problems. If you want to master these, then you should consider the following steps: Make sure that DIM and DEF FN and determine whether these commands are used in an

improper way. DEF FN should be at the beginning of the program (before the use of the function) and DIM must not be used to define array variables (see "Changes in Commodore Basic V2" under 2.4.6 "problem cases").

Adjust then determines whether the program has any memory, for example for machine programs or reloaded for a changed character set or the like. A sure sign of this is a POKE command, which has the address 56 (Basic memory end) to the destination. This command is usually followed by the CLR command. The POKE command you wish to remove, and instead use the compiler directive "£HEAPEND address" which directly indicates the Boss Basic, Basic end memory. With it you can ensure that the compiled program can be the reserved area untouched. You should also remove all other POKE commands that affect the addresses 43-56. If the compiled remaining space is not sufficient (the compiler reports "heapend unter heapstart" ("heapend below heapstart")), then you need to install either the reserved area or a part of the compilation (program area, range of variables or string field) to another location (possibly is "£ALLRAM" necessary). In any case, you should make sure that no POKE command accesses the program on one of the areas of the compiler.

If the Basic program is very long, you can precaution "£ALLRAM" use. However, it is important to ensure that the entire memory is then used. For example, if a machine program from $C000 (49152) is located, then you should set the basic memory end with "£HEAPEND $C000" before this area. If SYS commands are appended with parameters in the program, then "£OTHERON" is necessary. However, it must ensure that the machinery routines do not try to change the passed variable, but we also its value. Also, not all machine (language?) programs tolerate Basic-Boss. This helps best try. Unlike other Basic languages are not just assistance to structured programming; this is further complicated by the rather uncomfortable line numbers and the concept of global, two-digit variable. In addition, many consider it Basic programmers with the order not too sure why a basic program can easily degenerate into an uncontrollable chaos. Therefore must be considered in the adaptation in particular the following:

A Basic program normally operates exclusively in the Real type. However, since the compiled program for calculations in this type is not overly fast, you would certainly like to use the remaining types of Basic-Boss. This one should do but very economical, as in a Basic program is not so easy to determine where the variables are used. Therefore, it is advisable to redefine only a few variables that are used at central and especially significant for the speed locations. You have to know exactly where these variables are used throughout to ensure that they do not elsewhere must be of type Real, while you declare it as a Word, or Byte Integer. The code in this way is indeed quite long, but it can usually cope with ("£ALLRAM"). For this work, a cross-reference program can be very useful in indicating exactly where the variables are used.

Every now and then also provides the FOR-NEXT loop constitutes a problem. When popped in a subroutine of a loop with GOTO, then you risk in Basic only a stack overflow, the compiled program crashes against the next RETURN (see under "Changes in Commodore Basic V2" under 2.6.4 "problem cases"). Preventing you can do this by not jumping with a mere GOTO out of the loop, but set the loop variable first to the final value, execute a NEXT command, which does not lead to the beginning of the loop, so you can now work freely with GOTO. On this theme, you will find a program called "FORRETURN" on the Basic-Boss disk.

2.10 It does not work

If your program makes difficulties, the following reasons are possible:

- The program is faulty in itself
- The data types are applied incorrectly
- The program does not tolerate the characteristics of the compiler

If the first alternative is correct, I can not help you. Then you must proceed according to the general rules for the eradication of mistakes. But if you suspect you have to apply the data types is incorrect, you can tentatively set all variables to Real, by making all

declarations ineffective (with "£IGNORE" and "£USE") or by using the "@" after the REM remove commands. The compilation is then indeed quite long and inefficient, but if it has enough memory (if need, you can "£ALLRAM" to help take) and then runs the program, so this is an indication that any variable should be of type Real, but this is not. Now you can find the error, you can either conclude from the reactions of the program or in groups of redefining the variables back to the desired type. If the program will not run, is probably to blame for a peculiarity of the compiler. Is probably jumped out of a FOR-NEXT loop, so the compiled during the subsequent RETURN crashes. It is also possible that the Basic program calls machinery routines that are not compatible with the compiled program. Here is an overlap of memory regions is conceivable that the compiled program is often longer than the original. For this purpose you should look at the 2.6.4-"problem cases" under "Changes in Commodore Basic V2". Finally, there is an error in the Basic-Boss into consideration. Such is possible because the Basic-Boss is still very young. In such a case, you should ensure that there is actually a compiler error. The program should therefore run flawlessly under the interpreter and the errors described above should be largely excluded. I would be most obliged if you could isolate the problem as much as possible, that is, the faulty program should be made small as possible, without the error disappears. Then you should send me the faulty program with a precise description of the error (the error effects); very, very small programs than the expression and everything beyond that to disk. You may forget the version number of your compiler in any case.

2.11 Terms of use

The compilation contains routines and algorithms written by me, especially in the region routine. Nevertheless, I claim no copyright on the compiled program, provided these routines are not removed and used for other purposes. The compilation can thus be freely copied or sold unless the author of the compiled Basic program permits. In him alone the copyright for the compiled program is located. However, I think it is fair that is mentioned in the program or instructions that it was compiled with the Basic-Boss. For the compiler itself full copyright applies. It also applies to the compiler manual. Both can not (even partially) be copied or reproduced without written permission in any way. For damages caused by the compiler is incidentally not assume any liability or legal responsibility.

2.12 Error Messages

There are two types of error messages. The reports of a Basic-Boss already at compile time, and the others appear only in the execution of compilation.

2.12.1 The messages of the Basic-Boss

These messages appear during the compilation process. Since it is the Basic-Boss is a two-pass compiler, most error messages appear twice on (once in Pass 1 and once in Pass 2). The errors message consists of at least two rows. In the first line line number, and behind it a sort of extract the line are shown. This extract consists of the arithmetic operators used, a point for each instruction and the colon, which separate the commands. The extract allowed a more precise localization of the error. At least the double of points give an accurate information on the location of the error and the operators can be useful. If unknown operators appear occasionally (for example "k,g,v,u ....") so you should not worry about it, since this arises from an internal representation in the Basic-Boss. The line below contains the error number followed by the actual error message. The error number easier to find in the error list. There are also some errors that may never appear and the text makes sense only to me. Namely, if no error number is displayed and the error starts with a pound sign, then this indicates a problem in the Basic-Boss that really should not exist. In such cases, a division of a complex expression into smaller expressions fix the problem. A pound error can also occur when certain errors are preceded that have the Basic-Boss suddenly pulled out from the command processing. In these cases, the problem will resolve with the correction of other errors by itself. One should not rigidify when an error occurs on the specified error, because often out different types of writing errors to the same error. Therefore, you should check with untraceable difficulties the general syntax of the fault location. In particular,

the previous expression should be examined closely because a mistake can put in it, which causes the compiler to assume the expression to end early, which can lead to a variety of error messages. For longer variable names you should make sure the fact that they do not contain basic commands, as this may also have multiple effects. If an error appears, I can not guarantee the functionality of the compilation. Some errors are so serious that the compilation process is aborted when they occur anyway. If the displayed error line 65535, the error occurred after processing the last line of the program.

The error list

The errors are classified by their number. This order does not purport to be useful, because the list is only meant for reference. Follow a more detailed error description of the problem and possible suggestions to correct it.

1/HEAPEND IST UNTER HEAPSTART [HEAPEND IS UNDER HEAPSTART]

The end of the string memory is less than its beginning. This is typically the case when the compilation is too long and does not fit in the basic memory, making it about the end of string memory (usually 40,960 or $A000) extends. Remedy is for example the "£ALLRAM" command and / or the laying of the string with memory "£HEAPSTART" and "£HEAPEND". The error also occurs when you put the compiled program into another memory area, for example, after $C000. Then it must also "£HEAPEND" are applied.

2/DIVISION DURCH NULL [DIVISION BY ZERO]

It is a calculation performed with direct numbers, which requires a division by "0" (for example "PRINT 2/0").

3/BEREICHSUEBERSCHREITUNG [AREA ON VIOLATION]

The number range is exceeded by Real at a direct calculation (for example "PRINT 10^50").

4/FEHLER BEI DER BERECHNUNG [ERRORS IN THE CALCULATION]

Computing any error has occurred in a direct calculation is not precisely known.

5/ABBRUCH MIT RESTORE [CANCEL WITH RESTORE]

During compilation <RESTORE> was pressed.

7/CODE VERSCHOBEN [CODE MOVED]

The generated code of the previous line in Pass 2 is compared with the one generated in Pass 1 is shorter or longer. This error should only occur if other errors are preceded (for example "Constant is not defined"). It disappears when the other errors are corrected. When it occurs alone, then a variable of the same value was assigned twice.

8/FALSCHER IF-ZWEIG [INCORRECT IF BRANCH]

If in the expression with IF appears GOTO and should appear THEN, then this error appears. But it can also be the expression of IF in a faulty manner, the compiler assumes that the term end early.

9/UNBEKANNTER TOKEN [UNKNOWN TOKEN]

This message if the Basic-Boss is a token (a coded Basic command), he does not know will appear. This is usually when compiling programs the case, which were created using a basic extension and contain commands that extension.

10/ES IST LONGIF NOTWENDIG [IT IS NECESSARY LONGIF]

This error can occur only in SHORTIF mode and indicates that the IF line for this mode requires too much memory. Remedy the "£LONGIF" command either in the program or just before the beginning of the defective line. Behind this line, you can get back to "£SHORTIF" switch (for automatic switching would need a third pass).

11/ZU VIELE PARAMETER [TOO MANY PARAMETERS]

Here the Basic-Boss suspects a command with too many parameters. The error comes about when the Basic-Boss believes to have arrived at the end of command, but then neither is a colon or a newline. A typical cause is for example the SYS command suffix parameter. Then the "£OTHERON" command is recommended.

12/UNZULAESSIGE ZUWEISUNG [ILLEGAL ASSIGNMENT]

It was for example, attempts to assign something to a system variable, although this is not allowed. This is for example, in the case of TI. ST enables one to assign something.

13/UNBEKANNTER BEFEHL (=) [UNKNOWN COMMAND (=)]

The compiler believes to edit a variable assignment, but is behind the suspected variable with an equal sign. Then usually there is a command that was misspelled and the editor of the interpreter has therefore not recognized as a token. But it can also be a long variable name that contains the basic command words, which is not allowed.

14/UNBEKANNTER ZUSATZBEFEHL [ADDITIONAL UNKNOWN COMMAND]

The compiler did not recognize the additional instruction behind the left arrow. Presumably it was written incorrectly or it is the command of a Basic extension.

15/ADRESSBEFEHL ZU SPAET [ADDRESS COMMAND TO LATE]

A command like "£ROUTSTART", "£PROGSTART" etc. is not at the beginning of the program, as it should be and therefore can not be utilized.

16/COMPILERDIREKTIVE UNBEKANNT [UNKNOWN COMPILER DIRECTIVE]

The compiler does not understand an initiated with a pound sign compiler directive. It is probably a typo.

17/DATENTYP ERWARTET [DATA TYPE EXPECTED]

The compiler has at this point expects a data type such as "Byte" or "Word". There is probably a typo.

18/ZU LANGER NAME [TOO LONG NAME]

The file name is too long and has been truncated to 20 characters. This error also occurs when "£SYSTEXT".

19/DATEINAME ERWARTET [FILE NAME EXPECTED]

The compiler expected at this point a quoted file name. Probably lack the quotes. This error can also occur when "£SYSTEXT".

20/ES WURDE BEREITS CODE ERZEUGT [IT HAS BEEN GENERATED CODE]

The Basic-Boss has found a directive that is not the beginning of a program, there should be but because they will not work properly (for example "£SETPREFERENCES").

21/FOR OHNE NEXT (FASTFOR) [WITHOUT FOR NEXT (FASTFOR)]

The error only occurs in FASTFOR mode and then only at the end of the program or the "£SLOWFOR" command. In both cases, all FOR loops must be completed. If they are not, then this error occurs. One should looking for a FOR command, yet not exist for the NEXT command.

22/GO OHNE TO [GO WITHOUT TO]

Instead of "GOTO" can also write "GO TO". However, when "GO" without writing "TO," the Basic-Boss complains.

23/UNBEKANNTE ZEILE [UNKNOWN LINE]

There was a line number used for which there is no line. So you should examine where you actually want to jump to appeal to either the previous or the next line then.

24/',' ODER ')' ERWARTET [',' OR ')' EXPECTED]

When DIM command either a comma or a closing parenthesis is expected behind each dimension specified. Presumably, the expression has been suggested to end early, for which any typographical errors might be responsible.

25/FALSCHE DIMENSIONSANZAHL [WRONG NUMBER DIMENSION]

When DIM command an array is defined, which was used in the previous program text. Besides, many dimensions must be used in each case the same. If this is not observed, then this error appears.

26/'=' ERWARTET ['=' EXPECT]

Behind the loop variable must be specified in a FOR "=" are the Basic-Boss can not find here. It is also a long variable name, which contains basic commands, which is not allowed.

27/'TO'ERWARTET ['TO' EXPECT]

At the FOR command was on the site of "TO" found something else. Possible is a typing error or an error in the assignment after "FOR" so that its end is suspected to be early.

28/NEXT OHNE FOR (FASTFOR) [NEXT WITHOUT FOR (FASTFOR)]

It was tried in FASTFOR mode, complete a loop with NEXT, which does not exist. The reason for this could be, for example, that the "£FASTFOR" command in the program behind the FOR command is targeted instead of before it. In any case, you should examine the FOR-NEXT structure.

30/UNERLAUBTER ZAEHLERTYP [ILLEGAL COUNTER TYPE]

With FOR as may counter variable only numeric variables are used and no string variables. Arrays are also prohibited.

31/KOMMA ERWARTET [COMMA EXPECTED]

The Basic-Boss expects a comma at this point. However, it may also be that the end of the previous expression is thought too soon. It should be in doubt, check the entire expression here.

32/NACH DEM INPUTSTRING MUSS ';' FOLGEN [AFTER THE INPUT STRING ';' MUST FOLLOW]

If after the INPUT command is a parenthesized string in quotes, a must for this string ";" follow. After that, to read in variable follows.

33/NACH ON MUSS GOSUB ODER GOTO FOLGEN [MUST BE AFTER ON GOTO OR GOSUB FOLLOW]

After the ON command is followed by an expression, which in turn GOTO or GOSUB. If this order is not complied with, this error. It is also an error in the expression after ON, the result is that the phrase was supposed to end early.

34/DAS ARRAY WURDE BEREITS DIMENSIONIERT [THE ARRAY HAS ALREADY DIMENSIONED]

It tries to dimension an array twice with DIM was. One should look for the other DIM statement and eliminate.

35/FALSCHE NEXT-VARIABLE (FASTFOR) [WRONG NEXT VARIABLE (FASTFOR)]

This error occurs only in FASTFOR mode. It tries to complete a FOR loop with "NEXT X", but the specified variable at NEXT is not identical with the specified with FOR. Either the variable has been misspelled or the FOR-NEXT structure is messed up.

36/BEI DER FELDDIMENSIONIERUNG DUERFEN KEINE VARIABLEN [IN THE FIELD SIZING PERMISSIONS ARE NO VARIABLE UNRELATED]

If you dimension an array with "DIM A(B)", then you get this error, because the Basic-Boss can not handle variable sizing. So you have to use constant numbers for example "DIM A(1000)". If the field size before starting the program is not known, one has to dimension the field for better or worse to the largest possible size and may utilize memory losses. (See the "Changes in Commodore Basic V2" in Chapter 2.6.4 "problem cases".)

38/NUMMER ERWARTET [EXPECTED NUMBER]

It was expected a number between 0 and 65535. It may be used at this point is not a variable.

39/KLAMMER ZU FEHLT [MISSING BRACKET]

It is missing a closing parenthesis in the expression, or the expression is supposed to end early. It is also a excess opening bracket.

40/ZAHL ERWARTET [EXPECTED NUMBER]

It was expected a number. A string, a variable, or an arithmetic expression at this point trigger the error. This error can, for example appear in DATA when the data type was chosen numerically (for example "£DATATYPE Byte").

41/KLAMMER AUF FEHLT [MISSING OPEN BRACKET]

It lacks an opening parenthesis in expression, or a closing parenthesis is excess.

42/ZU VIELE FUNKTIONSPARAMETER [TOO MANY FUNCTION PARAMETERS]

It tried too many send a function parameter. It is also a missing closing parenthesis for nested functions and function arrays.

43/DAS KOMMA MUSS INNERHALB EINER FUNKTIONS/ARRAY-KLAMMERUNG STEHEN [THE COMMA MUST BE WITHIN PARENTHESES IN FUNCTION / ARRAY]

The Basic-Boss can do anything with a comma occurring in the expression. This error can occur only within a parenthesis.

44/STRING HIER NICHT ERLAUBT [STRING IS NOT ALLOWED]

The error is raised when a string expression is used in a function, although a numeric expression is required (for example "PRINT ABS(A$)").

45/ZUWENIG FUNKTIONSPARAMETER [INSUFFICIENT FUNCTION PARAMETERS]

Were given too little of a function parameter. It is also a superfluous closing parenthesis.

46/ZUVIEL ARRAYPARAMETER [TOO MUCH ARRAY PARAMETERS]

It attempts to address more dimensions of an array, as this has, according to its definition with DIM (the first use of an array of Basic-Boss also considered as a definition) was. It is also possible a missing closing parenthesis for nested functions and arrays.

47/ZUWENIG ARRAYPARAMETER [INSUFFICIENT ARRAY PARAMETERS]

Were specified with the use of array dimensions less than this, according to their own definition with DIM (the first use of an array of Basic-Boss also considered as a definition). It is also possible an excess close parenthesis for nested functions and arrays.

48/DAS ARRAY MUSS MINDESTENS EINDIMENSIONAL SEIN [THE ARRAY MUST BE AT LEAST ONE-DIMENSIONAL]

If less than one parameter is specified at the first use of an array, then this error appears.

49/UNERWARTETES AUSDRUCKENDE [UNEXPECTED END OF EXPRESSION]

If an expression abruptly breaks off, then you get this message. The Basic-Boss expected here may still be a number.

50/DEKLARATION ZU SPAET [DECLARATION TO LATE]

Declaring a variable using a compiler directive (for example "£WORD A,B") occurred only after the code has been generated. Therefore, the declaration of these variables can not be considered. It should therefore be at the beginning of the program as possible.

51/EIN STRING KANN NICHT IN EINE ZAHL UMGEWANDELT WERDEN [A STRING CAN NOT BE CONVERTED INTO A NUMBER]

It is a string expression was used at a point where a numeric expression should be.

52/EINE ZAHL KANN NICHT IN EINEN STRING UMGEWANDELT WERDEN [ONE NUMBER CAN NOT BE CONVERTED INTO A STRING]

It was used a numeric expression to a point where a string expression should be. May have been trying to link a numeric expression to a string.

53/FUNKTIONSPARAMETER MUESSEN IN KLAMMERN EINGESCHLOSSEN SEIN [FUNCTION PARAMETERS MUST BE ENCLOSED IN BRACKETS]

Behind a function is missing the opening parenthesis that there must necessarily be ("A = SIN 10" is not possible). So basically you can not do without the grip.

54/DEKLARATION EINER SYSTEMVARIABLEN [DECLARATION OF SYSTEM VARIABLES]

An attempt to declare a system variable (such as TI, TI$, ST), which is useless but because they have already been declared from the outset was.

55/VARIABLE ERWARTET [VARIABLE EXPECTED]

At this point, a variable or possibly a number was expected but not found.

57/DER SPEICHER IST ZU KNAPP [MEMORY IS TO SMALL]

The Basic-Boss is so expected its internal memory for variables, line numbers and constants. This should rarely be the case, since the Basic-Boss from version 2.2 it is very efficient with the main memory use.

58/DIMENSION UEBERSCHRITTEN [DIMENSION EXCEEDED]

When using constants as a parameter of an array variable of Basic-Boss can immediately determine whether these parameters exceed the engineered range. If, for example "DIM A(15)" is in the program and elsewhere "PRINT A(20)" is used, this error appears.

60/DISKFEHLER:... [DISK ERROR:...]

The floppy drive has the compiler signals an error. Behind "DISK ERROR:" You will receive the message that was read from the floppy. You can look up their meaning in floppy manual. After a disk error occurred, the compilation is aborted.

61/'FAST'ERWARTET ['FAST' EXPECTED]

If, in the declaration of a variable follows behind an equals sign, then it must follow that either a number or "FAST". Otherwise, this error appears (see "Data Types" in 2.3.2 "The Declaration").

62/ZUVIEL'FAST'-VARIABLEN [TOO MUCH 'FAST' VARIABLES]

A maximum of four bytes with FAST variables are assigned. This corresponds to two Word/Integer variables or four Byte variables. If these four bytes are used, then this error appears.

63/VERBOTENER DATENTYP [PROHIBITED DATA TYPE]

Only variables of type Byte, Word and Integer variables are declared as FAST. These can also be no arrays. The error can also occur if you try to assign a string variable, an address, which is also prohibited. Possibly also an array should be declared as a constant.

64/FUNKTIONSDEFINITION FALSCH [FUNCTION DEFINITION WRONG]

It has made an error in the definition of a FN function with "DEF". Maybe you should look at the syntax of the function definition again in the Basic Manual.

65/FUNKTION MEHRFACH DEFINIERT [FUNCTION DEFINED MULTIPLE]

The same FN function was defined more than once in the program with "DEF". The interpreter will not permit this, the Basic-Boss.

66/FUNKTION NICHT DEFINIERT [FUNCTION NOT DEFINED]

An attempt to apply an FN function was not defined at this time with "DEF" was. Such a function must be defined in a smaller line number, and prior to their application.

67/VERBOTENER TYP BEI FN [PROHIBITED IN TYPE FN]

If a FN function was found, the parameter is a string, then this message is displayed - for example, when "FN("abc")".

68/KONSTANTE BEREITS DEFINIERT [CONSTANT ALREADY DEFINED]

It attempts to assign a variable of type Constant has a value twice. However, this is allowed throughout the program only once.

69/KONSTANTE NICHT DEFINIERT [CONSTANT NOT DEFINED]

It is a variable of the data type constant was used, although it has not yet been defined in a previous line. See under "Data Types" chapter 2.3.10 "Data Type Constant".

70/FUNKTION/KONSTANTE ERWARTET [FUNCTION / CONSTANT EXPECTED]

An attempt is made to define a FN function or a constant, although the function of the text or constant text is missing. There is therefore a serious syntax error at "DEF FN" or the constant allocation.

71/COMPILERFUNKTION UNBEKANNT [COMPILER FUNCTION UNKNOWN]

The compiler with a "£" initiated function found that he does not know. Usually a typing error is the cause. See also Section 2.5 "New commands and functions".

2.12.2 The messages of compilation

These are the error messages that are issued during the execution of the compiled program. The number of the affected line is not normally displayed. This shortcoming can fix it if you used "£LINEON" (see under "£LINEON" in Chapter 2.4.8 "Other directives". The errors are not numbered, which is not really a problem as there are not very many, because most errors are already caught by the compiler. It used to be the normal English error messages from the interpreter. They usually have the same meaning as in the interpreter.

The Error List

NEXT WITHOUT FOR ERROR

The program has arrived at a NEXT command without previously a FOR command has been processed. You should carefully check the program run, therefore, to determine when this NEXT is called. This error can not occur in a FASTFOR loop.

OUT OF DATA ERROR

The READ command has run out of data. So you should check out when and where the READ command is executed and the data to which it should normally access.

TYPE MISMATCH ERROR

This error has the compiled program usually has a different meaning than the interpreter. It indicates that the data to be read with the READ command from a different type than the variable into which they are to be read. For example, has declared all data "£DATATYPE Byte" as byte data, and later the command "READ R" is executed (R be a floating point variable), then this error appears. If you still want to read it into a floating-point variable data, it must first be read into a Byte variable, then assign it to a Real variables (for example "READ B: R=B", where B is to be a Byte variable). If the data are of type String, this error normally occurs because the READ command strings can transform into any other type. But it will still appear when the conversion is impossible (you can not convert any letter string to a number).

ERROR BAD SUBSCRIPT

It attempts to address an array Elernent (element?) that does not exist was. The field limits were exceeded so if for example, as with "DIM A(100)" dimensioned a field and then with this "I=101:A(I)=1", then you get this error. By default, this error occurs but not in all fields. In one-dimensional Word, Integer or Byte fields, a field overrun is not noticed unless you use "SLOW ARRAY".

OUT OF MEMORY ERROR

If the string is not enough memory, then this error appears. You must then either increase the memory string (with "HEAPSTART" and "HEAPEND" or "ALLRAM", see chapter 2.4 "Description of the directives"), to deal with the strings or something more economical.

CAN'T CONTINUE ERROR

The error has nothing in common with its original meaning (it can normally only occur in direct mode). The compilation (more precisely, the garbage collection) shows him that the string memory is defective. The reason for this can be either a careless poke in the string memory or at worst an error in the string management of the compiler.

STRING TOO LONG ERROR

The error can only occur when adding strings (for example "A$=B$+C$"). The string to be added, have a total length of more than 255 characters, which is not processed my string management.

ILLEGAL QUANTITY ERROR

This error can be triggered in the MID$ function, if the second parameter is "0" (for example "MID$ (A$, 0, 2)"). It must be at least "1".

SYNTAX ERROR

If extended SYS commands are allowed "£OTHERON", then this error can also occur. It is triggered by the compiled code when the parameter passing is mixed up to the called machine program. In "£OTHERON" other faults may occur that causes the called machine program.

There are also other possible messages that originate from the Basic interpreter of C64/C128. These messages generally have the same meaning as usual (for example, "division by zero error" or "overflow error"). They can occur because the arithmetic routines of the Basic ROM be used by partially compiled code (for real numbers).

- Appendix A - Overview of all directives -

£ALLRAM	Turns on the use of total memory (62 Kbytes)
£BASICRAM	Turns off the use of total memory (62 Kbytes)
£BOOLEAN	Sets the variable type BOOLEAN (two states) established
£BOOLEANTYPE	Determines the expected type for IF-THEN
£BOSSOFF	Turns off Basic-Boss directives
£BOSSON	Turns on Basic-Boss directives
£BYTE	Specifies the type of variable Byte established (256 states)
£CALCTYPE	Determines the minimum type of all calculations
£CODE	Adds the code in the compiled program
£DATAFILE	Specifies the name of the device, and data files
£DATASTART	Specifies the start address of the data part of code
£DATATYPE	Specifies the type of the DATA data
£FASTARRAY	Turns the quick but dangerous field requests
£FASTFOR	Turns on the fast FOR-NEXT loop
£HEAPEND	Specifies the end address of the memory string
£HEAPSTART	Determines the starting address of the string memory
£HELPSTART	Specifies the start address of the memory help
£IGNORE	ignores all subsequent commands
£INITOFF	Turns off the initialization of the variable to "0"
£INITON	Enables the initialization of the variable to "0"
£INTEGER	Sets the variable type 'Integer' established
£LINEOFF	Turn off row update
£LINEON	Turn on row update
£LONGIF	Normal use IF statement

```
£LONGNAME      Switch to Basic-Boss variable detection
£OTHER         Allows unknown commands from Basic Extensions
£OTHEROFF      Prohibits additional parameter of the SYS command
£OTHERON       Allows additional parameters of the SYS command
£PRINTTYPE     Specifies the minimum variable type in PRINT statements
£PROGFILE      Specifies the device and file name of the program established
£PROGSTART     Specifies the start address of the program
£PROTOCOL     Causes the output of error messages and other data to the printer or to a file
£RAM           Switches the fast RAM access for PEEK and POKE in ALLRAM mode
£REAL          Sets the variable type "Real" established
£ROM           Turns on ALLRAM mode fast RAM access for PEEK and POKE from
£ROUTFILE      Determines the device and file name of the routines
£ROUTSTART     Sets start address that part of the routine
£SETPREFERENCES Raise the current settings to the default settings permanent
£SHORTIF       Use fast IF statement
£SHORTNAME     Turns on two digit variables recognition
£SLOWARRAY     Turns back to normal array accesses
£SLOWFOR       normal use FOR-NEXT loop
£SOURCEFILE    Specifies name, unit and secondary address of the source file
£STRING        Sets the variable type "String" established
£SYSNUMBER     Sets the line number of the SYS-line
£SYSOFF        Turns off SYS-line from the compilation
£SYSON         Turns on SYS-line from the compilation
£SYSTEXT       Sets the text in the SYS-line established
£USE           deactivation of "£IGNORE"
£VARSTART      Specifies start address that the variable memory
£WORD          Sets the variable type "Word" established


-------------------------------------------------
- Appendix B - Additional commands and functions -
-------------------------------------------------

<-BYTE      Writes a byte to the current output
<-CLI       Allows interrupts
<-IN        Specifies the input channel to the specified file
<-INTEGER   Writes two bytes to the current output
<-IOROM     Turns off I/O range and Basic ROM
<-LOAD      Loads a file into memory and executes the next command
<-OUT       Sets the output channel to the specified file
<-RAM       Switch off the ROM
<-REAL      Writes five bytes to the current output
<-RESET     Sets the output channel on the screen and the input channel on the keyboard
<-ROM       Turns on the ROM
<-SEI       Prohibits interrupts
<-WORD      Writes two bytes to the current output

@ BYTE      Reads a byte from the current output
@ INTEGER   Reads two bytes from the current output
@ REAL      Reads five bytes from the current output
@ WORD      Reads two bytes from the current output
```

Appendix C - Automatic mode

Importance of memory addresses:

1000,1001	They contain the values "123" and "234", the automatic mode is activated
1002	Device number source file
1003	Secondary address
1004	Length of file name
1005-1020	Filename
900	Device number target file
901	Secondary address
902	Length of file name
903-918	Filename

Appendix D - value ranges of the variable types

Type	Range minimum	Range maximum	Memory
Real	+-2.93873588*10(exp)-39 to	+-1.70141183*10(exp)38	5 Byte
Integer	-32768 to	+32767	2 Byte
Word	0 to	65535	2 Byte
Byte	0 to	255	1 Byte
Boolean	0 and	-1	1 Byte

Annex E - The Software Load Speeder Ultra Plus

The loading of Basic-Boss takes without floppy speeder approximately two minutes. If you are compiling multiple programs, so there is a considerable loading time. With the program "Ultraload Plus" described here, you can shorten the charging time by a multiple. The Basic-Boss, your Basic programs, and your compilations are then loaded into a seven-time speed. Please download the file:

```
load "ultraload plus",8
run
```

Now you can use the increased charging speed. Their programs invite as usual. The following description of "Ultraload Plus" is divided into two parts: The first part describes the program for the user (the benefits and opportunities), while the second part applies to interested programmers who want to know more about this system. But first, a simple description of this program. Even though you might not have your floppy for the C64 (C128) long, so be sure you have the 1541/1570/1571 have met the infamous long wait times. There are now several solutions to deal with the slow speed of the floppy disk drive, be it hardware or software. Ultrload Plus is a software solutions and, moreover, a very good one. The following are the benefits to you are listed as a user:

- Programs are loaded 6.8 times faster than normal.
- Directory entries are found much faster.
- Also, the command "VERIFY" will run much faster.
- Since the read-write head of the floppy is moved more than three times as fast as normal,

  is prevented thus become misaligned.
- You can interrupt with <RESTORE> the load anytime.

Maybe you already know the program "Hypra-Load". But Ultraload Plus is also better than Hypra-Load, as the following summary shows.

- Loading is quicker than Hypra-Load (6.8 times instead of 6.2 times).
- Hypra-Load does not accelerate the search of directory entries.
- The probability of a crash is lower on Ultra Plus than in Hypra-Load.
- With the help of several parameters can be easily adapted to Ultraload Plus programs not with the first floppy speeders (the term for such acceleration programs) worked together.
- Ultraload Plus is available in a read-write errors from a corresponding message without - as Hypra-Load - crashing.
- With Ultraload Plus the screen is not turned off.

So you see that Ultraload Plus brings several advantages. Above all you will but the blindingly fast strike. Gone are the coffee breaks when loading the 193 block long Basic-Boss. To collaborate with Ultraload Plus, you only need to start it with the commands mentioned above. Then you have the faster loading routines immediately. In the last line of the startup message you see a SYS command (SYS 336). You can use it, you should hold <RUN-STOP/RESTORE> or have triggered a reset. Even then you can continue to use the faster loading routines immediately, without having to reload the program.

Should there be a program that does not run smoothly with the floppy speeder, I would refer you to the following part, do not be put off by any special terms. In between, you will always find valuable information to work with this program, for example increasing the speed to up to 8 times! In addition to two tool (tools) provided, which are also described below. The fast data transfer from the disk drive to the computer is based on the simultaneous transmission of two bits on the restricted handshake mode as well as the time saving use of a table. In search of the Directory Entries own GCR encoding is used (Group Code Recording, as is the recording process of the 1541). The rapid data transmission to the drive is responsible for the delayed fractions of seconds starting the engine and is based on a very short transfer routine. Loading the directories (LOAD "$", 8 ...) is not accelerated. This is the VERIFY command - as mentioned above - processed faster. In contrast to the original routine-Verify, the program breaks when a fault occurs immediately. Use the command sequence "PRINT PEEK(174) + 256 * PEEK(175)" You can find the first different byte.

If you list the program directly after loading, you will see the following line:

1985 sys 2080,00288,192,214,n,3

In the following we will explain each parameter in detail. We will replace it but first through letters to make them easier to describe. Accordingly, the start line is as follows:

1985 sys 2080, a, b, c, d, e

A: Start address of the boot program
B: High byte of the starting address of the work area, must be equal to the following number on the two-part version and are from 16 to 200
C: High byte of the start address of the main part; must be on the two-part version of from 16 to 200, wherein the three-part from 16 to 246
D: Transfer speed (N for normal, H for high)
E: Reservation type memory (2 for two-parts, 3 for three-parts)

In principle, the length of the Basic line must always be the same. Therefore, any changes must be provided with leading zeros (eg 00288 instead of 288).

But now we come finally to the parameters: Ultraload Plus can be either two-part or three-part version in memory. For each part there exists a start address (A-C). Because two parts

of the address of the third program block can be omitted, it is necessary in the case B with the specified information coincide C (E must be 2).

A is simply given as a decimal, so it can take values between 0 and 65535 (these are all possible addresses in C64). It is recommended, however, to remain at the address 288. The following information about the data B and C are understood only assembly language programmers of you. But that's no reason to despair, there still below a solution for all the other readers will be offered. The boot routine (from 288) calls the main-program (about 2 Kbytes) on. The data for C (main part), determine by specifying each of the high byte of the address (decimal). So that means that the number 88 represents the start address 22528 (= $5800). As already mentioned, must correspond with the two-part version B and C.

This second part contains all loading routines and a small memory. It is protected against overwriting by itself, that is, it can not be destroyed when loading. This part of the program for example you can put in the RAM under the kernel or to the RAM under the I/O modules. The third and last part is the so-called work area (determined by B). This area is protected against overwriting. First, I would like to illustrate this principle, the memory layout of an example:

```
1985 sys 2080, 00288, 192, 214, n, 3
```

The boot part is therefore from address 288, the work area from 49152 (= $C000), the main part is from 54784 (= $D600), and the "3" at the end once again indicates that it is a three-part version. The fifth parameter specifies the transfer speed is to be loaded and saved with the. This "N" stands for normal and "H" for high. Would you like to improve the speed by a factor of 8, enter the following lines:

```
open 1,8,15
print# 1, "m-w"+chr$(105)+chr$(0)+chr$(1)+chr$(7);
close l
```

With this brief command sequence of the block spacing is changed on disk of usually ten to seven. This allows access Ultraload Plus more quickly to the disk. All programs that have been saved are also loaded later faster. However, this mechanism only works if the transfer mode to "H" have provided. Finally, I would like to address in the event that a program is not cooperating with Ultraload. Download the program to "ULTRALOAD TOOL 1", and run it. It fills the memory with a specific code, and then performs a reset. Download then your "problem child," and solve again a reset (SYS 64738 or, if necessary with a reset switch). Download now "ULTRALOAD TOOL 2", and then also this tool. Then you start all the possible addresses for the individual program components of Ultraload Plus display. These values can then immediately take over in the SYS command from Ultraload Plus.

(Thilo Herrmann / wb / ti)

```
5 REM *** LIES MICH !!! ***
15 :
50 PRINT CHR$(14)
100 PRINT"{clear}{down}{space*5}Dear Customer!"
200 PRINT"{down*2}{space*2}On the disk you will find the"
300 PRINT"{space*2}example programs that are"
400 PRINT"{space*2}mentioned in the manual."
500 PRINT"{down}{space*2}'AUTO.SHORT' is only a short"
600 PRINT"{space*2}version of 'AUTOBOSS', which"
700 PRINT"{space*2}can be easily incorporated"
800 PRINT"{space*2}into your own programs."
900 PRINT"{down}{space*2}There are also a few demo"
1000 PRINT"{space*2}programs for Basic-Boss."
1100 PRINT"{down}{space*2}Most were not so well adapted"
1200 PRINT"{space*2}to Basic-Boss, which is why the"
1300 PRINT"{space*2}variables are not very efficient,"
1400 PRINT"{space*2}so these programs are a bit long"
1500 PRINT"{space*2}and slow."
1600 PRINT
1700 PRINT"{down}{space*8}-SHIFT-":WAIT653,3
1800 PRINT"{clear}{down*2}
{space*2}'WAYOUT' is not too fast, because"
1900 PRINT"{space*2}it massively works with PRINT."
2000 PRINT"{space*2}'LABYRINTH' is too slow because"
2100 PRINT"{space*2}of unnecessary multiplications"
2200 PRINT"{space*2}and divisions."
2300 PRINT"{space*2}'BREAK' and 'DEMO' are optimally"
2400 PRINT"{space*2}fast and not very long."
2500 PRINT"{space*2}'FASTDIR' is from the 64'er,"
2600 PRINT"{space*2}and will not run on many"
2700 PRINT"{space*2}other compilers."
2800 PRINT"{space*2}'BOSSDIR' takes better advantage"
2900 PRINT"{space*2}of Basic-Boss."
3000 PRINT"{space*2}'RAMLOAD' can download programs"
3100 PRINT"{space*2}even after $D000, which is"
3200 PRINT"{space*2}illustrated with 'RAMDIR'."
3300 PRINT"{down*2}{space*8}-SHIFT-":WAIT653,3
3310 PRINT"{clear}{down*2}{space*6}Caution !"
3320 PRINT"{down}{space*2}The Floppy 1541 is not always"
3325 PRINT"{space*2}completely accurate. There is"
3330 PRINT"{space*2}a known bug when saving with"
3340 PRINT"{space*2}'spider', which was avoided"
3350 PRINT"{space*2}in Basic-Boss."
3360 PRINT"{down}{space*5}With The SCRATCH command, the"
3365 PRINT"{space*2}1541 does not always work"
3370 PRINT"{space*2}correctly, I think this happens"
3375 PRINT"{space*2}when another file is open. This"
3377 PRINT"{space*2}case is largely avoided in version"
3380 PRINT"{space*2}2.4 of the Basic-Boss. Only in one"
3382 PRINT"{space*2}case this is impossible: If the"
3384 PRINT"{space*2}error list is stored to the disk."
3386 PRINT"{space*2}Therefore you should be very"
3388 PRINT"{space*2}careful with this function."
3390 PRINT
3395 PRINT"{down}{space*8}-SHIFT-":WAIT653,3
3400 PRINT"{clear}{down*3}{space*2}I hope that otherwise"
3410 PRINT"{space*2}no major problems arise."
3500 PRINT"{down}{space*2}With kind regards"
3600 PRINT"{down}{space*17}Thilo Herrmann"
3610 PRINT"{down*2}{space*8}-SHIFT-":WAIT653,3
3620 PRINT "{clear}":PRINT CHR$(142)
```

```
8900 REM *** AUTO.SHORT ***
8909 :
8910 REM THIS SUBROUTINE IS A PACKED VERSION OF 'AUTOBOSS'.
8920 REM IT CAN EASILY BE INTEGRATED ON THE SCREEN EDITOR
8930 REM IN OTHER PROGRAMS!
8999 :
9000 REM@ £IGNORE
9030 S$="prog":L$="+prog":OPEN15,8,15:PRINT#15,"s:/"+S$
9040 PRINT#15,"r:/"+S$+"="+S$
9050 CLOSE15:SAVES$,8:OPEN15,8,15:INPUT#15,A,B$,C,D:IFATHENPRINTA;B$;C;D:END
9150 POKE1000,123:POKE1001,234:POKE1002,8:POKE1003,0:POKE1004,LEN(S$)
9160 FORI=1TOLEN(S$)
9170 POKE1004+I,ASC(MID$(S$,I,1)):NEXTI:POKE900,8:POKE901,0:POKE902,LEN(L$)
9180 FORI=1TOLEN(L$):POKE902+I,ASC(MID$(L$,I,1)):NEXTI:LOAD"basic-boss",8
9190 REM@ £USE
```

```
1 GOTO10
2 RUN9000
3 REM *** AUTOBOSS ***
4 REM JUST INFORMATION, IGNORE
5 REM@ £IGNORE
10 PRINT"{clear}{down}{ct n} This is a demo of the automatic"
11 PRINT" function of Basic-Boss."
12 PRINT" The routine starting at line 9000"
13 PRINT" can be used in any Basic program."
14 PRINT"{down} When it is called with RUN9000,"
15 PRINT" then this Basic program will be saved"
16 PRINT" (the old version is retained),"
17 PRINT" the compiler loads, compiles"
18 PRINT" the Basic program, then load"
19 PRINT" the compiled program and"
20 PRINT" started with RUN."
21 PRINT"{down} The name of the source program is"
22 PRINT" in s$ and the name of final program is"
23 PRINT" in l$ (it does not have to be"
24 PRINT" the Compiler program loaded)."
25 PRINT"{down} To try it, type 'run2'"
26 PRINT" (The Basic-Boss program must be"
27 PRINT" saved on the inserted disk.)"
90 END
98 :
99 REM NOW COMES THE PROGRAM
100 REM@ £USE
110 PRINT"{ct n}{down}This compiled program was"
120 PRINT"created automatically."
125 PRINT"{down*2} Quit with Run-Stop/Restore!"
129 REM VISUAL EFFECTS
130 REM@ {arrow left}SEI
140 POKE53280,1:POKE53280,0
150 POKE53280,3:GOTO140
900 END
8998 :
8999 :
9000 REM AUXILIARY ROUTINE IS NOT COPIED INTO THE COMPILED PROGRAM
9010 REM@ £IGNORE
9020 :
9030 REM NAME OF THE BASIC PROGRAM
9040 S$="prog"
9050 REM NAME OF THE FINAL PROGRAM
9060 L$="+prog"
9070 :
9080 OPEN15,8,15
9090 REM OLD FILE DELETE
9100 PRINT#15,"s:/"+S$
9110 REM OLD FILE RENAME
9120 PRINT#15,"r:/"+S$+"="+S$
9130 CLOSE15
9131 REM ACTUAL PROGRAM SAVE
9132 SAVE S$,8
9134 OPEN15,8,15
9136 INPUT#15,A,B$,C,D
9137 IF A THEN PRINTA;B$;C;D:END
9140 :
9150 REM AUTOMATIC MODE
9160 POKE1000,123:POKE1001,234
9170 :
9180 REM DEFINITION OF THE SOURCE FILE
9190 POKE1002,8  :REM DEVICE
9200 POKE1003,0  :REM SECONDARY ADDRESS
```

```
9210 POKE1004,LEN(S$) :REM NAME LENGTH
9220 FORI=1TOLEN(S$)  :REM NAME
9230 POKE1004+I,ASC(MID$(S$,I,1)):NEXTI
9240 :
9250 REM DEFINITION OF THE FINAL PROGRAM
9260 POKE900,8    :REM DEVICE
9270 POKE901,0    :REM SECONDARY ADDRESS
9280 POKE902,LEN(L$)  :REM NAME LENGTH
9290 FORI=1TOLEN(L$)  :REM NAME
9300 POKE902+I,ASC(MID$(L$,I,1)):NEXTI
9310 :
9311 REM BASICBOSS LOADING
9312 LOAD"basic-boss",8
9315 REM REST BACK COMPILATION
9320 REM@ £USE
```

```
3 REM *** BIGARRAY ***
4 :
5 REM@ £ALLRAM:£FASTFOR:£SHORTIF
10 REM@ £BYTE A(:£WORD I=FAST
20 DIM A(62000)
30 FOR I=0 TO 62000
40 A(I)=I AND $FF
50 NEXT I
55 PRINT"data inserted into array."
60 FOR I=0 TO 62000
70 IF A(I)<>(I AND $FF) THEN PRINT"error"
80 NEXT I
85 PRINT"test done."
```

```
1 REM *** BOSSDATEI ***
2 :
3 REM@ £FASTFOR:£WORD I=FAST
10 PRINT"{down*2} this is a demonstration of the"
20 PRINT" new file commands !{down*2}"
29 REM DELETE OLD FILE
30 OPEN 1,8,15,"s:testfile":CLOSE 1
38 :
39 REM OPEN NEW FILE
40 OPEN 1,8,2,"testfile,s,w"
50 {arrow left}OUT 1:REM OUTPUT ON # 1 (CMD 1)
59 REM WRITING DIFFERENT VALUES
60 FOR I=1024 TO 2023:{arrow left}BYTE PEEK(I)
70 POKEI,160:NEXT I
80 {arrow left}REAL {pi}
90 {arrow left}WORD 12345
100 {arrow left}RESET:REM OUTPUT TO SCREEN
110 CLOSE 1
120 :
139 REM OPEN OLD FILE
140 OPEN 1,8,2,"testfile,s,r"
150 {arrow left}IN 1:REM ENTERING #1
159 REM READ AGAIN VALUES
170 FOR I=1024 TO 2023:POKEI,@BYTE
180 NEXT I
190 A=@REAL:PRINT A
200 PRINT @WORD
210 {arrow left}RESET:REM INPUT FROM KEYBOARD
220 CLOSE 1
```

```
1 REM *** BOSSDIR ***
3 :
5 REM@ £BYTE #:£WORD X
10 OPEN1,8,0,"$"
20 {arrow left}IN 1
25 X=@WORD:REM GET LOAD ADDRESS
30 X=@WORD:REM GET LINE LINK
35 IF ST THEN 70
40 PRINT @WORD;:REM DISPLAY BLOCK NUMBER
50 A=@BYTE:{arrow left}BYTE A:IF A THEN 50
60 PRINT:GOTO30
70 {arrow left}RESET
80 CLOSE 1
```

```
5 REM *** BREAK ***
12 :
20 REM@ £WORD #:£FASTFOR:£SHORTIF
30 REM@ £BYTE X,SO,J,I2,IE,COUNT,LV,SL
40 POKE650,128
500 SO=15:BR=10:B2=BR-1
900 AP=1023:GOSUB5750
950 GOSUB5700
1000 JO=56320
1005 GOSUB12000
1010 GOSUB10000
1020 GOSUB5100:REM CALCULATE
1030 GOSUB5200:REM TEST
1032 GOSUB8000:REM KEYBOARD
1035 POKEAP,32:
1040 POKEPO,81:
1060 GOSUB 5000
1070 FORI=1TOSP:NEXT
1080 GOSUB 5000
1085 FORI=1TOSP:NEXT
1090 GOTO1020
5000 RI=0
5005 IF (PEEK(JO)AND4)=0 THEN 5030:REM JOYSTICK
5010 IF (PEEK(JO)AND8)=0 THEN 5040
5020 RETURN
5030 IFXC>1984THENXC=XC-1:RI=-128
5032 POKEXC,119:POKEXC+BR,32:RETURN
5040 IFXC<1984+40-BRTHEN XC=XC+1:RI=128
5042 POKEXC+B2,119:POKEXC-1,32:RETURN
5090 :
5100 REM CALCULATE
5110 AX=X:AP=PO:PO=PO+DY
5120 XX=XX+DX:IFXXAND256THENXX=XXAND255:PO=PO+TX:X=X+TX
5130 REMPRINT"{home}{space*30}{home}";PO;X;XX;DX;TX
5140 RETURN
5190 :
5200 PN=PO+DY
5210 IF PEEK(PO)=204 OR PEEK(PO)=239 THEN GOSUB5600
5230 IF PO<1104THEN DY=40:GOSUB6000
5240 IF X=39 THEN IF TX=1 THEN DX=-DX:TX=-TX:GOSUB6000
5250 IF X=0 THEN IF TX=-1 THEN DX=-DX:TX=-TX:GOSUB6000
5255 IF PO>=1944THEN GOSUB 5270
5260 RETURN
5270 IF PEEK(PN)=119 THENGOTO5500
5275 IF PO>=1988THEN POKEAP,32:GOSUB5700:DX=0:TX=0:DY=0:IFSL=1THENGOSUB5950
5280 RETURN
5399 :
5500 DY=-40:GOSUB6200
5505 IF X=0AND TX=0 THEN DX=128:TX=1
5506 IF X=39AND TX=0 THEN DX=-128:TX=-1
5510 DX=DX+RI:TX=0
5515 IF DX+30000>30000THENTX=1
5520 IF DX+30000<30000THENTX=-1
5525 IF DX+30256<30000THENTX=-1:DX=-256::RETURN
5530 IF DX>256THEN IF DX<1000THEN DX=256:TX=1:RETURN
5590 RETURN
5600 DY=-DY:IFPO>=1104THENGOSUB6100
5610 P=(PO-X)+(XAND255-3):POKEP,32:POKEP+1,32:POKEP+2,32:POKEP+3,32
5630 COUNT=COUNT-1:PU=PU+1:PRINT"{home}"TAB(9)PU
5640 IFCOUNT>0THENRETURN
5650 SL=SL+1:PRINT"{home}{down}{right}
next level...":GOSUB5800:GOSUB10000:GOSUB5700
5660 LV=LV+1
5690 RETURN
5700 DY=-40:DX=256:TX=1:X=5:PO=1904+X:AP=1023
5710 XX=X*256:RETURN
5750 XC=1984+20:IF BR>19 THEN XC=1984
5760 IE=3:LV=1:SL=3:PU=0
5790 RETURN
5800 POKE198,0
5810 GETA$:IFA$=""THEN5810
```

```
5820 RETURN
5900 PRINT"{home}{gray}{space*2}score: {left}"PU"{left}{space*4}
level:"LV"{left}{space*4}live: "SL"{left}{space*2}":RETURN
5950 PRINT"{home}{down} end ":GOSUB5800
5960 GOSUB5700:GOSUB5750:GOSUB10000:RETURN
6000 :
6100 :
6200 :
6510 POKE54272+24,SO:SO=15-SO
6530 RETURN
6999 :
8000 IF PEEK(198)=0THEN RETURN
8005 GETA$
8010 IFA$="+"ANDBR<40THENBR=BR+1:IFXC>1984THENXC=XC-1
8020 IFA$="-"ANDBR>1THENBR=BR-1
8100 IF A$>="0" AND A$<="9" THEN GOSUB9000
8800 B2=BR-1:GOSUB11000
8850 IFDY<>0THENRETURN
8860 GOSUB5700:SL=SL-1:GOSUB5900
8870 AP=1023:LV=LV+1
8900 RETURN
9000 A=9-VAL(A$)
9010 SP=(2^A-1)*7+200*A:RETURN
10000 POKE53281,12:PRINT"{gray}{clear}{white}";
10002 GOSUB5900:PRINT"{home}"
10005 A$(1)="{white}":A$(2)="{yellow}":A$(3)="{orange}":A$(4)="{red}"
10006 IE=IE+2:IFIE>15THENIE=15
10007 COUNT=IE*7
10010 FORI2=1TOIE:PRINTA$((I2AND3)+1);
10020 J=INT(RND(1)*3)*4
10030 PRINTTAB(J)"{reverse on}L{cm p*3}L{cm p*3}L{cm p*3}L{cm p*3}L{cm p*3}
L{cm p*3}L{cm p*3}{reverse off}"
10040 IF (I2AND3)=3 THEN PRINT
10100 NEXTI2:PRINT"{gray}";
10150 POKE53281,0:POKE53280,12
11000 FORI=1984TO2023:POKEI,32:NEXT
11010 FORI=0TOB2:POKEXC+I,119:NEXT
11020 RETURN
11110 FORI=0TOB2:POKE1984+XC+I,119:NEXT
12000 PRINT"{clear}{ct h}"CHR$(142)
12010 PRINT"{space*10}simple breakout"
12020 PRINT"{space*3}a demo for basic-boss compiler"
12022 PRINT"{down}{space*14}(thilo herrmann, 1988)"
12023 PRINT
12025 PRINT"{down}{space*4}this games can (and should)"
12026 PRINT"{space*4}be freely copied."
12029 PRINT"{down}{space*4}operation during the game:"
12030 PRINT"{down}{space*2}+,- ... width"
12040 PRINT"{space*2}0-9 ... speed"
12042 PRINT"{down*2}{space*2}the direction of move of the paddle"
12043 PRINT"{space*2}influenced the bouncing off the ball."
12044 PRINT"{down}{space*2}please choose the"
12045 PRINT"{space*2}start speed: (0-9)"
12047 PRINT"{down} (for the basic version choose '9')"
12050 GETA$:IFA$=""THEN12050
12060 GOTO9000
```

```
3 REM *** DEMO ***
6 :
10 REM@ £FASTFOR:£SHORTIF:£DATATYPE BYTE
12 REM@ £WORD #,I=FAST,X=FAST
13 REM@ £CONSTANT VIC,BORD,BACKG,SPREN,SCOL,XHI,IRQ,OFF,VOL,MEM
15 REM@ £BYTE A,X1,X2,Y1,Y2,XA,XE,YA,YE,CO,CC,SX,SY,V1,H1,V2,H2,Y(,YS
(,CO(
16 REM@ £BYTE P2(,SN
17 DIM SC(24),CO(23),X(7),Y(7),XS(7),YS(7),P2(7)
20 VIC=53248:BRDR=VIC+32:BACKG=VIC+33:SPREN=VIC+21:OFF=VIC+17:IRQ=56333
21 XHI=VIC+16:SCOL=VIC+39:PRI=VIC+27:MEM=2040
22 VOL=54272+24
28 GOSUB 13000:REM PREPARATION
29 POKE BRDR,0:POKE BACKG,0:POKE SPREN,0
30 PRINT"{white}{ct h}{ct n}{clear}{space*7}*** Basic-Boss ***"
32 PRINT"{down}Programming in Basic with the"
34 PRINT"speed of machine language!"
40 PRINT"{down}{gray}Impossible? No!"
42 PRINT"This program was programmed"
44 PRINT"from beginning to end in Basic."
50 PRINT"Then it was compiled by Basic-Boss"
52 PRINT"in pure machine language."
60 PRINT"{down}Please convince yourself of its"
70 PRINT"speed!"
90 GOTO 20000
100 PRINT"{clear}{down*2}{space*2}How it looks when the screen"
110 PRINT"{down}{space*2}color is changed in rapid"
120 PRINT"{down}{space*2}succession:":GOSUB 10020
130 POKE IRQ,127:REM INTERRUPT OFF
135 POKE OFF,0:REM SCREEN OFF
150 FOR I=0 TO 60000
180 POKE BRDR,14
190 POKE BRDR,3
200 POKE BRDR,3
210 POKE BRDR,14
220 POKE BRDR,6
240 NEXT I
250 POKE IRQ,129:POKE OFF,27
253 PRINT"{clear}{down*2} or:":GOSUB 10020
265 POKE IRQ,127:REM INTERRUPT OFF
266 POKE OFF,0:REM SCREEN OFF
270 FOR I=0 TO 30000
271 POKE BRDR,0
272 POKE BRDR,2
273 POKE BRDR,2
274 POKE BRDR,2
275 POKE BRDR,8
276 POKE BRDR,8
277 POKE BRDR,8
280 POKE BRDR,7
281 POKE BRDR,1
282 POKE BRDR,1
283 POKE BRDR,7
284 POKE BRDR,8
285 POKE BRDR,8
286 POKE BRDR,8
287 POKE BRDR,2
288 POKE BRDR,2
289 POKE BRDR,2
290 POKE BRDR,2
291 POKE BRDR,0
295 NEXT
300 POKE IRQ,129:POKE OFF,27:RETURN
305 :
310 PRINT"{clear}{down*2}{space*2}When a Basic program is compiled"
320 PRINT"{down}{space*2}with Basic-Boss the processed"
330 PRINT"{down}{space*2}screen looks like this:"
```

```
340 PRINT"{down}{space*2}"
350 GOSUB 10020:X1=10:X2=1:Y1=14:Y2=4
355 GOSUB 11000
360 I=0
370 X1=X1+33:IF X1>=40 THEN X1=X1-40
375 X2=X2+17:IF X2>=40 THEN X2=X2-40
380 Y1=Y1+21:IF Y1>=25 THEN Y1=Y1-25
385 Y2=Y2+7 :IF Y2>=25 THEN Y2=Y2-25
390 CO=(CO+1 AND 15)
395 GOSUB 12000
400 I=I+1
405 IF I<1000 AND PEEK(198)=0 THEN 370
410 MU=11:GOSUB 14000:PRINT"{home}{down*2} Or:"
415 GOSUB 10020
420 GOSUB 11000:H1=1:V1=2:H2=2:V2=1:I=0
425 X1=1:Y1=2:X2=37:Y2=22:CC=1
430 IF X1 =0  THEN H1=-H1
440 IF X2<=1  THEN H2=-H2
450 IF Y1 =0  THEN V1=-V1
460 IF Y2 =0  THEN V2=-V2
470 IF X1 =39 THEN H1=-H1
480 IF X2>=38 THEN H2=-H2
490 IF Y1 =24 THEN V1=-V1
500 IF Y2 =24 THEN V2=-V2
510 X1=X1+H1:X2=X2+H2
520 Y1=Y1+V1:Y2=Y2+V2
525 CO=CO(CC):CC=CC+1:IF CC>23 THEN CC=0
530 GOSUB 12000
540 I=I+1:IF I<1000 AND PEEK(198)=0 THEN 430
550 MU=500:GOSUB 14000:RETURN
560 :
570 PRINT"{clear}Now bounce some sprites over the"
580 PRINT"{down}screen. However, here there it is a"
590 PRINT"{down}problem: the program is too fast."
600 PRINT"{down}It must therefore be slowed down:"
605 PRINT"{down}":GOTO 1000
610 :
620 FOR I=0 TO 7
630 POKE MEM+I,13
640 POKE SCOL+I,I+1
645 X(I)=130+I*25:Y(I)=50+I*18
647 XS(I)=-I:YS(I)=I
650 NEXT I
660 POKE PRI,0:POKE SPREN,255
662 A=0:C=0
665 :
670 FOR I=0 TO 7
690 IF X(I) AND 256 THEN POKE XHI,PEEK(XHI) OR P2(I):GOTO 710
700 POKE XHI,PEEK(XHI) AND (255-P2(I))
710 POKE VIC+I+I,X(I) AND 255
715 POKE VIC+1+I+I,Y(I)
720 X(I)=X(I)+XS(I)
730 Y(I)=Y(I)+YS(I)
740 IF X(I)>320 THEN X(I)=640-X(I):XS(I)=-XS(I):GOSUB 950
750 IF X(I)<24 THEN X(I)=48-X(I):XS(I)=-XS(I):GOSUB 950
760 IF Y(I)<50 THEN Y(I)=100-Y(I):YS(I)=-YS(I):GOSUB 950
770 IF Y(I)>229 THEN Y(I)=458-Y(I):YS(I)=-YS(I):GOSUB 950
800 REM ACCELERATION X AND Y
810 IF A<3 THEN 880
820 XS(I)=XS(I)-1:YS(I)=YS(I)+1
880 NEXT I
882 IFA=3 THEN A=0
883 A=A+1
886 REM WAITING RASTER BEAM
887 IF B THEN IF (PEEK(53248+17)AND 128)=0 THEN 887
890 IF PEEK(198)=0 THEN 670
900 POKE198,0:  RETURN
```

```
950 POKE VOL,SN:SN=15-SN:RETURN
990 END
999 :
1000 GOSUB 10600:B=0:GOSUB 610
1010 PRINT"{down*2}now it is slowed down and it is"
1020 PRINT"{down*2}synchronized with the raster beam."
1030 GOSUB 10600
1050 B=-1:GOSUB 610
1090 RETURN
9999 :
10000 TI$="000000":GOTO 10100
10010 TI$="000030":GOTO 10100
10020 TI$="000035":GOTO 10100
10100 GOSUB 10600:GOTO 10500
10500 POKE 198,0
10510 GET A$:IF A$="" AND TI$<"000040" THEN 10510
10520 RETURN
10600 PRINT"{down*3}{space*10}- Press any key -":RETURN
10998 :
11000 FOR I=1024 TO 2023:POKE I,160:NEXT
11010 RETURN
11997 :
11998 REM RECTANGLE DRAWING WITH COLOR
11999 REM (X1,Y1,X2,Y2,CH,CO)
12000 IF X2>=X1 THEN XA=X1:XE=X2:GOTO 12002
12001 XA=X2:XE=X1
12002 IF Y2>=Y1 THEN YA=Y1:YE=Y2:GOTO 12050
12003 YA=Y2:YE=Y1
12050 FOR Y=SC(YA) TO SC(YE) STEP 40
12060 FOR X=Y+XA TO Y+XE
12070 POKE X,CO:NEXT X,Y
12090 RETURN
12998 :
12999 REM MULTIPLICATION TABLE
13000 FOR I=0 TO 24:SC(I)=55296+I*40:NEXT
13010 REM COLOR READING
13020 FOR A=0 TO 23:READ CO(A):NEXT
13030 MP=0
13040 FOR A=0 TO 7:P2(A)=2^A:NEXT
13050 FOR I=832 TO 832+62
13060 READ A:POKE I,A:NEXT I
13090 RETURN
13499 REM COLOR DATA
13500 DATA 0,6,14,3,1,3,14,6,0,2,8,7,1,7,8,2,0,11,5,13,1,13,5,11
13599 REM SPRITE DATA
13600 DATA   0,255,  0,  3,255,192, 15
13601 DATA 255,240, 31,255,248, 63,255
13602 DATA 252,127,255,254,127,255,254
13603 DATA 255,255,255,255,255,255,255
13604 DATA 255,255,255,255,255,255,255
13605 DATA 255,255,255,255,255,255,255
13606 DATA 127,255,254,127,255,254, 63
13607 DATA 255,252, 31,255,248, 15,255
13608 DATA 240,  3,255,192,  0,255,  0
13998 :
13999 REM CLEAR SCREEN(MU)
14000 I=1024:A=21
14010 FOR A=1 TO 5:NEXT A
14020 POKE I,32:I=I+MU
14030 IF I>=2045 THEN I=I-1021
14040 IF I<>1024 THEN 14010
14050 RETURN
20000 :
20010 PRINT"{down}{right*2}Please select:"
20020 PRINT"{down}{space*3}1...Screen demo"
20030 PRINT"{space*3}2...Sprite demo"
20040 PRINT"{space*3}3...Background demo"
```

```
20050 PRINT"{space*3}4...Something else"
20090 PRINT"{down*2}(Thilo Herrmann, 1988)"
20092 :
20094 :
20100 TI$="000000":GOSUB 10500
20110 IF A$>="1" AND A$<="4" THEN MP=VAL(A$):GOTO 20130
20120 IF A$<>"" THEN 20100
20125 MP=MP+1:IF MP>4 THEN MP=1
20130 ON MP GOSUB 310,570,100,21000
20140 GOTO 29
21000 PRINT"{clear}{down*2}For speed comparison, you should run"
21010 PRINT"the Basic version of this program!"
21020 PRINT"{down}The Basic program also shows that"
21030 PRINT"the programmers can exploit all"
21040 PRINT"possibilities of Basic without being"
21050 PRINT"restricted."
21060 PRINT"{down}In addition to these screen effects,"
21070 PRINT"more serious applications can be"
21080 PRINT"programmed, for example Basic-Boss"
21090 PRINT"has a more advanced management of"
21100 PRINT"Strings that the Basic interpreter."
21110 PRINT"Therefore, the Garbage-Collection"
21120 PRINT"is now faster."
21130 GOSUB 10000
21140 PRINT"{clear}You will notice what this means when"
21150 PRINT"you run the following program:{down}"
21160 PRINT
21170 PRINT"10 dim a$(2000)"
21180 PRINT"20 for i=1 to 2000"
21190 PRINT"30 a$(i)=chr$(65):next i"
21200 PRINT"40 ti$="000000":print"CHR$(34)"free"CHR$(34)"fre(0);ti/60"
21210 PRINT"{down}For example, the FRE command takes 339"
21220 PRINT"seconds because it triggers a Garbage-"
21230 PRINT"Collection."
21240 PRINT"{down}You can now run the same program"
21250 PRINT"in the compiled version: ":GOSUB 10000
21260 PRINT"{clear}{down*2} starting..."
21300 DIM A$(2000)
21310 FOR I=1 TO 2000
21320 A$(I)=CHR$(65):NEXT I
21330 TI$="000000":PRINT"free"FRE(0);TI/60
21335 PRINT"{down*2}So, the garbage collection in this"
21336 PRINT"{down}case is about 680 times faster!"
21340 GOSUB 10010
21400 PRINT"{clear}{down*2}Features of Basic-Boss:"
21410 PRINT"{down}- Short compilations"
21420 PRINT"- Optimized and efficient code"
21430 PRINT"- Very fast variable types"
21440 PRINT"- Extremely short times to compile"
21450 PRINT"- A flexible compiler architecture"
21460 PRINT"- 62 Kbytes of Basic memory"
21470 PRINT"- Quick FOR-NEXT loop"
21480 PRINT"- Packed and fast data in DATA"
21490 PRINT"- Highly accelerated arrays"
21500 PRINT"- Any long variable names"
21510 PRINT"- Accurate German error messages"
21520 PRINT"- Generate real machine language code"
21525 PRINT"- No copy protection"
21530 PRINT"{down*2} and a lot more..."
21540 GOSUB 10000
21550 PRINT"{clear}{down*3} I am grateful to anyone who share"
21560 PRINT" {down}this demo program."
21570 GOTO 10000
```

```
1 REM *** ERRORREAD ***
3 :
5 REM@ £BYTE A
10 PRINT "{clear}":PRINT CHR$(14):PRINT"{down}{space*2}
{ct n}ASCII File Loader!"
15 PRINT"{down}{space*3}(runs only as a compiled program)"
20 PRINT"{space*3}Displays a text file"
30 PRINT" {sh space} to the screen!"
32 PRINT"{space*3}Press a button to stop"
33 PRINT" {sh space} the display."
60 INPUT"{down}{space*2}Text file:";T$
61 PRINT"{down}{space*2}SEQ or PRG (S/P) ?"
62 GETA$:IFA$="s"THEN65
63 IFA$<>"p"THEN62
65 :
67 PRINT
70 OPEN1,8,2,T$+","+A$:REM OPEN FILE
80 {arrow left}IN 1
90 {arrow left}BYTE @BYTE    :REM GET AND DISPLAY
99 REM BUTTON IS PRESSED?
100 IF PEEK(198) THEN GOSUB200
109 REM END OF FILE OR RUN / STOP?
110 IF ST=0 AND PEEK(197)<>63 THEN 90
115 :
120 {arrow left}RESET:CLOSE1
130 PRINT CHR$(142):END
190 :
199 REM WAIT, LET FLOPPY TIME OUT
200 {arrow left}RESET
210 POKE198,0:WAIT198,1:POKE198,0
220 {arrow left}IN1
230 RETURN
```

```
5 REM *** FASTDIR ***
15 :
1000 OPEN1,8,0,"$":POKE781,1:SYS65478:GETA$,A$
1010 GETA$,A$:IFST=64THENSYS65484:CLOSE1:END
1020 GETA$,B$:PRINT"{left}"ASC(A$+CHR$(0))+256*ASC
(B$+CHR$(0));
1030 GETA$:PRINTA$;:IFA$<>""THEN1030
1040 PRINT:GOTO1010
```

```
5 REM *** FASTMULDIV ***
8 :
10 REM@ £WORD #,I=FAST:£REAL R:£FASTFOR
15 E=10000:R=0.3333333333
20 PRINT"{clear}{down}{ct n}{space*8}Information"
30 PRINT"{down*2}{space*2}When a Word / Integer variable"
40 PRINT"{space*2}is multiplied by a power of two,"
50 PRINT"{space*2}then Basic-Boss executed the"
60 PRINT"{space*2}operation much faster than"
70 PRINT"{space*2}before version 2.4."
75 PRINT
80 PRINT"{space*2}Similarly, it is a division"
90 PRINT"{space*2}of Word variables."
100 PRINT"{down}{space*2}>>A*8<< is faster than >>A*9<<."
110 PRINT
120 PRINT"{down}{space*2}so 10000 times >>B=A*9<<"
122 PRINT"{space*2}takes a longer time:"
125 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
130 TI$="000000":A=1000:PRINT"{clear}{down}{space*2}Time:";
140 FORI=1TOE:B=A*9:NEXT
150 PRINTTI/60-R
160 PRINT"{down}{space*2}>>B=A*8<< is faster:"
165 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
170 PRINT"":TI$="000000":A=1000:PRINT"{down}{space*2}Time:";
180 FORI=1TOE:B=8*A:NEXT
190 PRINTTI/60-R
200 PRINT"{down}{space*2}>>A=A*8<< is even faster:"
205 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
210 TI$="000000":A=0:PRINT"{down}{space*2}Time:";
220 FORI=1TOE:A=8*A:NEXT
230 PRINTTI/60-R
240 PRINT"{down}{space*2}The most efficient is >>B=A*256<<"
242 PRINT"{space*2}compiled:"
245 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
250 TI$="000000":A=1000:PRINT"{clear}{down}{space*2}Time:";
260 FORI=1TOE:B=256*A:NEXT
270 PRINTTI/60-R
290 PRINT"{down}{space*2}Similarly, with >>B=A/257<<"
295 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
300 TI$="000000":A=1000:PRINT"{down}{space*2}Time:";
305 FORI=1TOE:B=A/257:NEXT
310 PRINTTI/60-R
315 PRINT"{down}{space*2}und >>B=A/256<<"
325 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
330 TI$="000000":A=1000:PRINT"{down}{space*2}Time:";
335 FORI=1TOE:B=A*256:NEXT
340 PRINTTI/60-R
350 PRINT"{down}{space*3}-PRESS ANY KEY-":POKE198,0:WAIT198,1
390 PRINT"{clear}{down}{space*2}A multiplication by 2^8=256"
400 PRINT"{space*2}is therefore the fastest."
410 PRINT"{space*2}{down}Furthermore: >>A*2<< is faster"
420 PRINT"{space*2}than >>A*128<<. >>A*512<< is"
430 PRINT"{space*2}faster than >>A*32768<<."
440 PRINT"{down}{space*2}The same applies to the Division."
```

```
5 REM *** FORRETURN ***
8 :
10 PRINT"{clear}{down*2}{ct n}{space*8}FOR and RETURN"
12 PRINT"{down*2}{space*5}If a FOR loop is not correctly"
14 PRINT"{space*2}terminated with NEXT, and"
16 PRINT"{space*2}the computer instead runs on"
18 PRINT"{space*2}a RETURN, then a system"
20 PRINT"{space*2}crash is the result!"
22 PRINT"{space*2}This occurs for example when"
24 PRINT"{space*2}you exit prematurely with"
26 PRINT"{space*2}RETURN or GOTO a FOR loop"
28 PRINT"{space*2}in a subroutine."
30 PRINT"{space*5}You can prevent this if you"
32 PRINT"{space*2}do not just jump out of the"
34 PRINT"{space*2}loop, but previously sets the"
36 PRINT"{space*2}loop variable by the NEXT"
38 PRINT"{space*2}final value, and only"
40 PRINT"{space*2}then executes a RETURN or"
42 PRINT"{space*2}GOTO."
44 PRINT"{down}{space*2}-press any key-":WAIT198,1:POKE198,0
46 PRINT"{clear}{down*2}{space*2}
Clean programming style isn't"
48 PRINT"{space*2}all that sure, which is why I"
50 PRINT"{space*2}recommend to solve the problem"
52 PRINT"{space*2}with IF and GOTO instead of"
54 PRINT"{space*2}FOR-NEXT."
56 PRINT"{down}{space*2}By the way, In {pound}FASTFOR loops"
58 PRINT"{space*2}this problem does not occur."
60 PRINT"{down}{space*2}There are three sample routines"
62 PRINT"{space*2}in line 300, 400 and 500. The"
64 PRINT"{space*2}first works correctly and the"
66 PRINT"{space*2}other two in basic too. They crash,"
68 PRINT"{space*2}however, in the compiled program."
70 PRINT"{down}{space*2}-press any key-":WAIT198,1:POKE198,0
72 PRINT"{clear}{down*2}{space*2}The first is called now!"
74 PRINT"{down}{space*2}-press any key-
{down}":WAIT198,1:POKE198,0
76 :
78 GOSUB 200
80 :
82 PRINT:PRINT"{down*2}{space*2}
Now the second is called, which"
84 PRINT"{space*2}sends the C-64 in the desert."
88 PRINT"{down}{space*2}-press any key-
{down}":WAIT198,1:POKE198,0
90 :
92 GOSUB 300
94 :
96 PRINT:PRINT"{down*2}{space*2}
That was probably the interpreter!"
98 END
196 :
197 :
198 REM THIS SUBROUTINE IS CORRECT!
199 :
200 FOR I=1 TO 10
210 PRINT I;
```

```
220 IF I>4 THEN I=100:NEXT I:RETURN
230 NEXT I
240 RETURN
296 :
297 :
298 REM THIS SUBROUTINE GENERATES A SYSTEM CRASH
299 :
300 FOR I=1 TO 10
310 PRINT I;
320 IF I>4 THEN RETURN
330 NEXT I
340 RETURN
396 :
397 :
398 REM THIS SUBROUTINE CREATES ALSO A SYSTEM CRASH
399 :
400 FOR I=1 TO 10
410 PRINT I;
420 IF I>4 THEN 440
430 NEXT I
440 RETURN
```

```
0 REM *** LABYRINTH ***
0 :
0 REM@ £INTEGER #
1 REM@ £WORD A,B,C,X,P,TC,BC,DW:£BYTE CO,Z,Y
2 POKE53281,0:POKE53280,6
3 S=54272:GOSUB7000:GOSUB9000
4 VP=54272:CO=1
5 GOTO900
9 REM@ £FASTFOR
10 Y=PEEK(TC-U):Z=PEEK(TC-U+LL):IFABS(U)=1ORDC=0THEN30
20 IFY=32ANDZ=32THENFORA=TC-U+LLTOBC-V-LLSTEP40:POKEA,W3:POKEA+VP,CO:NEXT
30 POKETC,TW:POKEBC,BW:POKETC+VP,CO:POKEBC+VP,CO
31 B=B+1:IFB<DWTHENTC=TC+U:BC=BC+V:GOTO30
40 IFABS(U)=1THENPOKETC,T1:POKEBC,B1:POKETC+VP,CO:POKEBC+VP,CO
50 IFTC=BC-LLTHENRETURN
60 FORB=TC+LLTOBC-LLSTEP40:POKEB,SW:POKEB+VP,5:NEXT:RETURN
65 DW=INT(DV/3)+1
70 F=ML:U=LL+1:V=1-LL:T1=80:B1=122:SW=103
80 FORX=4TO6STEP2:TW=77:BW=78:B=0:IFX=4THEN100
90 F=MR:U=LL-1:V=-LL-1:SW=101:T1=79:B1=76:TW=78:BW=77
100 LM=M(F):LD=W(F)
101 IFF=3ANDOP=ECTHEN140
104 IFLD=0THENLD=2
105 IFQ-LD*INT(Q/LD)=0THENTC=M(X):BC=M(X+1):GOSUB10:GOTO130
110 TW=99:BW=100:U=U-LL:V=V+LL:N=PEEK(OP+LM)
120 IFN-D*INT(N/D)=0THENTC=M(X)+DW*LL:BC=M(X+1)+DW*-LL:GOSUB10
130 IFABS(U)=1THENU=U+LL:V=V-LL
140 M(X)=M(X)+DW*U:M(X+1)=M(X+1)+DW*V:NEXT
150 IFM=3ANDOP=ECTHENRETURN
160 IFQ-D*INT(Q/D)=0THEN180
170 OP=OP+M(M):Q=PEEK(OP):DC=DC+1
172 IFDC>=CDTHENRETURN
173 ZX=ZX+1
174 DW=DV-ZX+1:GOTO70
175 IFDC=CDTHENRETURN
180 R=PEEK(M(4)-1)
185 IFR=32THENM(4)=M(4)-1:M(5)=M(5)-1:IFM(4)<>TL+DW*LLTHEN180
190 R=PEEK(M(6)+1)
195 IFR=32THENM(6)=M(6)+1:M(7)=M(7)+1:IFM(6)<>TR+DW*LLTHEN190
200 WK=1:FORTC=M(4)TOM(6):POKETC,W1:POKETC+VP,CO:NEXT
210 FORBC=M(5)TOM(7):POKEBC,W2:POKEBC+VP,CO:NEXT:RETURN
240 CX=ZX:ZX=0:GK=WK:WK=0
241 REM@ £SLOWFOR
242 FORI=0TO10:GETA$:NEXT
243 PRINT"{home}o2:"OX"{left} "
245 GOSUB15000
247 IFA$="p"THENGOSUB5000 :GOTO245
250 NP=P:Q=PEEK(P):IFA$="g"THENNP=P+M(M)
260 IFNP=PTHEN320
270 IFTT=1ANDCX=0ANDGK=0THENGOSUB6000:GOSUB4000:RUN
285 IFQ-D*INT(Q/D)<>0THENP=NP:GOTO380
305 POKES+13,15*16+12:POKES+7,90:POKES+8,4:POKES+11,129:POKES+11,128
320 NM=M:IFA$="r"THENNM=M+1
330 IFA$="o"THENNM=M+2
340 IFA$="l"THENNM=M+3
345 IFA$="m"THENGOSUB600:POKE198,0:WAIT198,1:POKE198,0:GOTO360
350 IFNM=MTHEN242
360 IFNM>3THENNM=NM-4*INT(NM/4)
370 M=NM
380 D=W(M):Q=PEEK(P):OP=P:ML=M-1:MR=M+1
390 IFML+30000<30000THENML=3
400 IFMR>3THENMR=0
410 OX=OX-OL
415 IFOX+30000<30000THENPRINT"air consumed!":GOSUB6800:GOSUB4000:RUN
420 T=T+1:TT=1
430 IFTT=1THENDC=0:GOSUB2000:GOSUB65
440 M(4)=TL:M(5)=BL:M(6)=TR:M(7)=BR:GOTO240
500 P=P+1:IFP>EMTHENP=SM
510 A=INT(RND(1)*4):DC=0
520 A=A+1:DC=DC+1:IFDC>3THEN500
530 IFA>3THENA=0
540 M=P+M(A):IFM<SMORM>EMTHEN520
550 CP=PEEK(P):CM=PEEK(M):IFC+10000>10000ANDCP=210THENP=M:GOTO510
560 TM=M-SM:IF(CP=CMORCM<210)ANDC+10000>10000THEN520
570 ME=TM-L*INT(TM/L):IF(ME=0ANDM(A)=1)OR(ME=GANDM(A)=-1)THEN520
580 OD=INT(15/W(A)):CP=CP/W(A):POKEP,CP:CM=CM/OD:POKEM,CM
590 PRINTTAB(17);H-C"{left}{space*5}{up}"
```

```
594 P=M:C=C+1:IFC<HTHEN510
595 RETURN
600 GOSUB2000
602 REM@ £FASTFOR
605 A=1025+LL:MS=A:DC=SM:DD=SM+G
610 FORB=DCTODD:FORC=0TO3:CB=PEEK(B):CB=CB-W(C)*INT(CB/W(C))
620 AA=1:IFC=1ORC=3THENAA=LL
630 IFC>1THENAA=-AA
640 BB=LL/AA:P2=A+AA:P1=P2+BB:P3=P2-BB
645 IFB=PTHENPOKEA,W4:POKEA+VP,CO
650 IFCB=0THENPOKEP1,W3:POKEP2,W3:POKEP3,W3
651 IFCB=0THENPOKEP1+VP,CO:POKEP2+VP,CO:POKEP3+VP,CO
660 NEXT:A=A+2:NEXT:DC=DC+L:DD=DD+L:A=MS+(2*LL):MS=A
670 IFDD<=EMTHEN610
680 GOSUB7000:RETURN
690 REM@ £SLOWFOR
900 PRINT"{clear}{down}{light blue}you hide in a maze on a planet and you
{space*2}have";
910 PRINT" to find a way out before you run{space*3}out of oxygen."
920 PRINT"{down}{right}{pink}use the joystick or:"
922 PRINT"{down}{right*4}l = left{space*32}r = right"
923 PRINT"{right*4}o = 180 degree rotation{right*17}g = go"
924 PRINT"{down}{right*4}{orange}p = printer{light blue}"
925 PRINT"{down}{right*4}{orange}m = map{light blue}"
930 PRINT"{down*3}{right*2}{gray}do want to enter the size (y/n)?"
932 PRINT"{down}{light green}{right*1}{orange}
otherwise through random calculations{down*2}"
934 GETA$:IFA$="n"THEN940
935 IFA$<>"y"THEN934
936 INPUT"{clear}{down*5}{right*3}{gray}length:";LZ:IFLZ<1ORLZ>12THEN936
937 INPUT"{down*6}{right*3}width:";BZ
938 IFBZ<1ORBZ>19THENPRINT"{up*8}":GOTO937
940 PRINT"{clear}{purple}{down*3}{right*3}input of the amount of oxygen"
941 PRINT"{right*3}{down*2}press return for calculation"
942 GX=0:INPUT"{white}{down*6}{right*3}oxygen:";GX
943 IFGX>999THEN940
945 PRINT"{clear}{down*10}{right*13}{white}please wait{orange}{down*3}"
950 TL=1032:BL=1992:TR=TL+24:BR=BL+24:LL=40:DV=4:CD=5
955 W1=99:W2=100:W3=160:W4=88
960 W(0)=5:W(1)=7:W(2)=3:W(3)=2:L=INT(RND(1)*5+6):IFBZ>0THENL=BZ
970 M(0)=1:M(1)=L:M(2)=-1:M(3)=-L:M(4)=TL:M(5)=BL:M(6)=TR:M(7)=BR
975 W=INT(RND(1)*5+6):IFLZ>0THENW=LZ
980 H=L*W-1:G=L-1:C=0:DC=0:T=0:RM=0
995 SM=25000:EM=SM+H:FORA=SMTOEM:POKEA,210:NEXT:P=SM+INT(RND(1)*H)
1000 EC=SM+INT(RND(1)*L):MC=INT(RND(1)*H)+SM
1003 IFOX>0THEN1010
1005 OX=INT(H/3)+1
1010 IFP<OXTHENOX=-OX
1015 OX=OX+H:OL=1:TT=INT(RND(1)*H/8)+1:GOSUB500:GOSUB2000
1017 IFGX>0THENOX=GX+1
1030 M=0:NP=P:CE=PEEK(EC):CE=CE/W(3):POKEEC,CE:GOTO380
2000 PRINT"{clear}{down*24}":RETURN
3000 FORX=0TO1000:PRINT"{up}":NEXT:RETURN
4000 GETA$:IFA$=""THEN4000
4010 POKE53281,11:CO=8:PRINT"{clear}{light blue}{down*5}{right*2}map(y/n)?"
4020 GETA$:IFA$="n"THEN4500
4030 IFA$<>"y"THEN4020
4040 PRINT"{clear}";:GOSUB605
4060 PRINT"{home}{down*23}{right*10}-press any key-"
4065 FORI=1TO11:GETA$:NEXT
4070 GETA$:IFA$=""THEN4070
4500 PRINT"{clear}{down*16}{right*4}{orange}one more time(y/n)?{light blue}"
4510 GETA$:IFA$="y"THENRETURN
4520 IFA$<>"n"THEN4510
4530 PRINT"{clear}";:POKE53280,14:POKE53281,6:END
5000 OPEN4,4:PRINT#4,CHR$(14)
5005 FORI=1024TO2023STEP40:P$=""
5010 FORJ=0TO39:X=PEEK(J+I)
5020 IFX<32THENX=X+64:GOTO5100
5025 IFX<64THEN5100
5030 IFX<96THENX=X+32:GOTO5100
5035 IFX<128THENX=X+64:GOTO5100
5100 P$=P$+CHR$(X)
5150 NEXT:PRINT#4,P$:NEXT
5155 CLOSE4
5160 RETURN
6000 :
```

```
6001 PRINT"{right*5}{white}exit reached"
6010 GOSUB3000
6720 RETURN
6800 POKES+13,15*16+12:POKES+11,129
6805 POKES+24,15+16:POKES+23,15*16+1:POKES+6,15*16:POKES,200:POKES+1,3
6810 POKES+4,129:FORI=3000TO0STEP-5:POKE53280,7:POKES+7,IAND255:POKES+8,I/256
6815 POKE53280,6:NEXT:POKES+1,1:POKES,170
6820 GOSUB8000:GOSUB7000:RETURN
7000 FORI=0TO24:POKES+I,0:NEXT:POKES+6,15*16+11:POKES+24,15:POKES,162:POKES+1,14
7010 RETURN
7500 POKES+23,2+15*16:POKES+24,15+16:RETURN
8000 PRINT"{clear}";
8002 A$="{orange}{white}{red}{cyan}{purple}{green}{blue}{yellow}{orange}{brown}
{pink}{dark gray}{gray}{light green}{light blue}
{light gray}":FORI=1TO35:PRINTMID$(A$,(IAND15)+1,1);:GOSUB8008
8003 NEXTI:RETURN
8008 PRINT"{home}{down*2}{right*5}";
8009 PRINT"{space*12}{cm @*4}{space*24}";
8010 PRINT"{space*11}N{space*3}N{cm g}{space*23}";
8020 PRINT"{space*11}O{cm y*2}P {cm g}{space*23}";
8030 PRINT"{space*8}{cm p*3}{cm g}{space*2}{cm m} L{cm p*3}{space*20}";
8040 PRINT"{space*7}N{space*3}{cm g}{space*2}{cm m}N{space*3}N{cm h}{space*19}";
8050 PRINT"{space*7}O{cm y*3}{space*4}{cm y*3}P {cm h}{space*19}";
8060 PRINT"{space*7}{cm h}{space*10}{cm n} {cm h}{space*19}";
8070 PRINT"{space*7}L{cm p*3}{space*4}{cm p*3}{sh @}N{space*20}";
8080 PRINT"{space*11}{cm h}{space*2}{cm n} {cm h}{space*23}";
8090 PRINT"{space*11}{cm h}{space*2}{cm n} {cm h}{space*23}";
8100 PRINT"{space*11}{cm h}{space*2}{cm n} {cm h}{space*23}";
8110 PRINT"{space*11}{cm h}{space*2}{cm n} {cm h}{space*23}";
8120 PRINT"{space*11}{cm h}{space*2}{cm n} {cm h}{space*23}";
8130 PRINT"{space*11}{cm h}{space*2}{cm n} {cm h}{space*23}";
8140 PRINT"{space*11}L{cm p*2}{sh @}N{space*24}";
8150 PRINT"{space*40}";
8160 PRINT"{space*40}";
8170 PRINT"{yellow}{space*4}{reverse on}{space*2}rest{space*2}in{space*2}peace
{space*2}"
8180 RETURN
9000 PRINT"{clear}{white}{down*14}{space*2}basic-boss compiler demo{down}"
9001 PRINT"{down} 'labyrinth' written entirely in basic!"
9002 PRINT
9004 PRINT"{down*2} (thilo herrmann, 1988)"
9006 PRINTCHR$(142)CHR$(8)
9007 GOTO9200
9008 PRINT"{home}{down*2}{right*4}";
9009 PRINT"{cm k}{space*39}";
9010 PRINT"{cm k}{space*8}{cm k}{space*17}{cm k}{space*2}{cm k}{space*9}";
9020 PRINT"{cm k}{space*8}{cm k}{space*11}S{space*5}{cm k}{space*2}{cm k}
{space*9}";
9030 PRINT"{cm k}{space*4}{cm i*3} {cm k}{space*16}{cm u}O{cm u} {cm k}
{space*9}";
9040 PRINT"{reverse on} {reverse off}{space*7}{cm k}O{cm u*2}M {cm k}{space*2}
{cm k}{cm l}N{cm v}{cm k}{cm l}{cm u}M{space*2}{cm k}{space*2}O{cm u}M
{space*7}";
9050 PRINT"{reverse on} {reverse off}{space*4}N{cm u*2}{cm k*2}{space*2}
{cm l} {cm k}{space*2}{cm k}{cm l}{space*2}{cm k}{cm l}{space*2}{cm k} {cm k}
{space*2}{cm k} {cm l}{space*7}";
9060 PRINT"{reverse on} {reverse off}{space*4}{cm k}{space*2}{cm k*2}{space*2}
N {cm k}{space*2}{cm k}{cm l}{space*2}{cm k}{cm l}{space*2}{cm k} {cm k}
{space*2}{cm k} {cm l}{space*7}";
9070 PRINT"{cm u*4} {cm u*3} {cm u*3}{space*2}{cm u*3}{cm k}{space*22}";
9080 PRINT"{space*17}{cm k}{space*22}";
9090 PRINT"{space*15}{cm i}N"
9095 RETURN
9200 A$="{white}{red}{cyan}{purple}{green}{blue}{yellow}{orange}{brown}{pink}
{dark gray}{gray}{light green}{light blue}{light gray}{orange}":I=0
9202 GETB$:IFB$<>""THENRETURN
9205 I=I+1:PRINTMID$(A$,(IAND15)+1,1);
9210 GOSUB9008:GOTO9202
9220 RETURN
15000 A$=""
15005 IF(PEEK(56320)AND1)=0THENA$="g"
15010 IF(PEEK(56320)AND2)=0THENA$="o"
15020 IF(PEEK(56320)AND4)=0THENA$="l"
15030 IF(PEEK(56320)AND8)=0THENA$="r"
15040 IF(PEEK(56320)AND16)=0THENA$="m"
15050 IF A$=""THENGETA$:GOTO15005
15090 RETURN
```

```
5 REM *** OTHER ***
8 :
10 REM@ {pound}OTHERON
20 GOTO21:£OTHER !COL
21 :
29 REM LOAD EXTENSION
30 IF PEEK(49153)<>11ORPEEK(49156)
<>3 THEN LOAD"newcom.o",8,1
34 REM START EXTENSION
35 SYS49152
40 PRINT CHR$(14):PRINT"{clear}{down*2}{ct n}{white}
{space*2}This is a demonstration of the"
50 PRINT"{space*2}directives £OTHERON and"
60 PRINT"{space*2}£OTHER. You allow the"
70 PRINT"{space*2}execution of the command"
80 PRINT"{space*2}>>!COL A,B<< from the sample"
90 PRINT"{space*2}extension. This extension is"
100 PRINT"{space*2}located under the name of"
110 PRINT"{space*2}'NEWCOM.O' on the floppy disk."
120 PRINT"{down}{space*2}It is started with >>SYS 49152<<."
130 PRINT"{down}{space*2}'NEWCOM.SRC' contains the source"
140 PRINT"{space*2}in text format."
150 PRINT"{down}{space*2}(It sets the screen colors)"
160 PRINT
170 PRINT"{space*9}-PRESS ANY KEY-"
180 POKE198,0:WAIT198,1:PRINT CHR$(142)
190 :
200 REM EXAMPLE OF COMMAND:
210 !COL 12,11
220 PRINT"{down}{space*2}OK"
```

```
4 REM *** PREFERENCES ***
5 :
6 REM ** EXAMPLE TO SETPREFERENCES **
7 :
8 REM FIRST THE DESIRED SETTINGS:
9 :
10 REM@ £WORD #
20 REM@ £FASTFOR:£FASTARRAY:£SHORTIF
30 REM@ £LONGNAME
40 REM@ £OTHERON
50 REM@ £ALLRAM
60 REM@ £SOURCEFILE "mybasic"
97 :
98 REM NOW THE SETTINGS ARE PERMANENTLY TRANSFERRED:
99 :
100 REM@ £SETPREFERENCES
101 :
110 REM THIS LINE IS NOT REACHED BY THE COMPILER!
119 PRINT CHR$(14)
120 PRINT"{clear}{down*2}{space*2}{ct n}
When you compile this program,"
130 PRINT"{space*2}a Basic-Boss is created which"
140 PRINT"{space*2}uses: Word as default variable"
150 PRINT"{space*2}type, fast loops, fast arrays,"
160 PRINT"{space*2}IF-branches, any length"
170 PRINT"{space*2}variables name, allows SYS"
190 PRINT"{space*2}parameters and use all"
200 PRINT"{space*2}the Basic memory."
210 PRINT"{space*2}"
220 :
225 PRINT"{space*2}Also 'MYBASIC' is considered"
226 PRINT"{space*2}source file by default."
227 :
230 PRINT"{down}{space*2}(Unless otherwise is"
240 PRINT"{space*3}set in the source code)"
250 PRINT"{down}{space*2}You can save this Basic-Boss"
260 PRINT"{space*2}pressing '{arrow left}' with"
270 PRINT"{space*2}'*MYBOSS' and use it in the"
280 PRINT"{space*2}future !"
290 WAIT198,1:PRINT CHR$(142)
```

```
0 REM *** RAMDIR ***
0 :
1 REM THIS SOURCE FILE IS COMPILED
2 REM IN 'R+RAMDIR' AND 'P+RAMDIR'.
3 REM
4 REM 'P+RAMDIR' CAN ONLY BE LOADED
5 REM WITH '+RAMLOAD' FILE!!!
6 :
7 :
8 REM@ £BYTE #:£WORD X
9 REM@ £ALLRAM:£PROGSTART $D000
10 OPEN1,8,0,"$"
20 {arrow left}IN 1
25 X=@WORD:REM GET LOADING ADDRESS
30 X=@WORD:REM GET LINK LINES !?
35 IF ST THEN 70
40 PRINT @WORD;:REM BLOCK NUMBER DISPLAY
50 A=@BYTE:{arrow left}BYTE A:IF A THEN 50
60 PRINT:GOTO30
70 {arrow left}RESET
80 CLOSE 1
```

```
5 REM *** RAMLOAD ***
15 :
100 REM@ £ALLRAM:£RAM:£WORD I=FAST
110 PRINT CHR$(14)
120 PRINT"{down}{ct n} RAMLOAD - loads the specified file"
130 PRINT" into the RAM - even under the I/O range"
140 PRINT" from $D000!"
150 PRINT"{down} So, for example, the file 'P+RAMDIR'"
170 PRINT" is loaded. If 'R + RAMDIR' load"
175 PRINT"  normally, the program also works"
180 PRINT" at $D000."
200 INPUT"{down} File name:";D$
210 OPEN 1,8,0,D$
220 {arrow left}IN 1
230 I=@WORD: REM GET START ADDRESS
240 PRINT"{down} Start address:"I
250 POKE I,@BYTE
260 I=I+1
270 IF ST=0 THEN 250
280 PRINT"{down}{space*6}until:"I
290 {arrow left}RESET:PRINT CHR$(142)
300 CLOSE 1
```

```
5 REM *** RAMROM ***
10 REM@ £WORD I=FAST:£ALLRAM:£FASTFOR
20 :
30 PRINT"{clear}{down*2} the graphics memory is"
40 PRINT" described in the rom mode:"
50 GOSUB500
60 GOSUB600
65 REM@ £ROM
70 FOR I=8192 TO I+7999:POKEI,255:NEXTI
75 GOSUB700
80 :
90 PRINT"{clear}{down*2} the graphics memory is"
100 PRINT" described in the ram mode:"
110 GOSUB500
120 GOSUB600
125 REM@ £RAM
130 FOR I=8192 TO I+7999:POKEI,0:NEXTI
132 REM IN THE INTEREST OF THE REST OF THE PROGRAM RETURN TO ROM MODE
135 REM@ £ROM
140 GOSUB700
150 PRINT"{down}finished !{down}"
490 END
498 :
499 :
500 PRINT"{down} - press any key -"
510 POKE198,0:WAIT198,1:POKE198,0:RETURN
590 :
591 REM FOLLOWING POKES INTO THE VIDEO
592 REM CHIP INSTEAD OF THE RAM
594 :
595 REM@ £ROM
598 :
599 REM GRAPHICS TURNING ON
600 POKE53248+17,59:POKE53248+24,24
610 PRINT"{clear}":RETURN
698 :
699 REM GRAPHICS SWITCHING OFF
700 POKE53248+17,27:POKE53248+24,21
710 PRINT"{clear}":RETURN
```

```
2 REM *** WAYOUT ***
4 :
5 REM@ £INTEGER #:£LONGIF
8 BS=1024:F=55296
33 POKE53280,0:POKE53281,0:PRINT"{142}{clear}{ct h}{down*4}
{yellow}":CLR
34 C0=0:C1=1:C2=2:C3=3:C4=4:C5=5:C6=6:C7=7:C8=8:C9=9
35 SC=1024:LL=40
36 DIMA(19,11),B(19,11)
37 A$(1)="north":A$(2)="east":A$(3)="south":A$(4)="west"
38 W$="{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}
{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}
{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}{cm e}{cm r}"
39 SP$="{space*47}"
40 M1=C0:PRINT"{reverse off}{green}{down*2}
enter the dimension of the labyrinth:"
41 INPUT"{down}how wide:";H
42 IFH=C0ORH>19THENPRINT"{up*2}";:GOTO41
43 INPUT"{down}how long";V
44 IFV=C0ORV>10THENPRINT"{up*2}";:GOTO43
45 PRINT"{down}now an"H"V"V"labyrinth is generated"
46 A=H*V+C1
47 Q=C0:Z=C0:X=INT(H*RND(1))+C1
48 AA=X
49 A(X,0)=C1:C=C2
50 R=X:S=C1:GOTO57
51 IFR<>HTHEN55
52 IFS<>VTHEN54
53 R=C1:S=C1:GOTO56
54 R=C1:S=S+C1:GOTO56
55 R=R+C1
56 IFA(R,S-C1)=C0THEN51
57 IFR-C1=C0THEN89
58 IFA(R-C1,S-C1)THEN89
59 IFS-C1=C0THEN73
60 IFA(R,S-C2)THEN73
61 IFR=HTHEN65
62 IFA(R+C1,S-C1)THEN65
63 X=INT(C3*RND(1))+C1
64 ONXGOTO120,124,128
65 IFS<>VTHEN68
66 IFZ=C1THEN71
67 Q=C1:GOTO69
68 IFA(R,S)THEN71
69 X=INT(C3*RND(1))+C1
70 ONXGOTO120,124,135
71 X=INT(C2*RND(1))+C1
72 GOTO70
73 IFR=HTHEN83
74 IFA(R+C1,S-C1)GOTO83
75 IFS<>VTHEN78
76 IFZ=C1THEN81
77 Q=C1:GOTO79
78 IFA(R,S)THEN81
79 X=INT(C3*RND(1))+C1
80 ONXGOTO120,128,135
81 X=INT(C2*RND(1))+C1
```

```
82 GOTO80
83 IFS<>VTHEN86
84 IFZ=C1THEN120
85 Q=C1:GOTO87
86 IFA(R,S)THEN120
87 X=INT(C2*RND(1))+C1
88 ONXGOTO120,135
89 IFS-C1=C0THEN107
90 IFA(R,S-C2)THEN107
91 IFR=HTHEN101
92 IFA(R+C1,S-C1)THEN101
93 IFS<>VTHEN96
94 IFZ=C1THEN99
95 Q=C1:GOTO97
96 IFA(R,S)THEN99
97 X=INT(C3*RND(1))+C1
98 ONXGOTO124,128,135
99 X=INT(C2*RND(1))+C1
100 GOTO98
101 IFS<>VTHEN104
102 IFZ=C1THEN124
103 Q=C1:GOTO105
104 IFA(R,S)THEN124
105 X=INT(C2*RND(1))+C1
106 ONXGOTO124,135
107 IFR=HTHEN115
108 IFA(R+C1,S-C1)THEN115
109 IFS<>VTHEN112
110 IFZ=C1THEN128
111 Q=C1:GOTO113
112 IFA(R,S)THEN128
113 X=INT(C2*RND(1))+C1
114 ONXGOTO128,135
115 IFS<>VTHEN118
116 IFZ=C1THEN51
117 Q=C1:GOTO119
118 IFA(R,S)THEN51
119 GOTO135
120 A(R-C1,S-C1)=C
121 C=C+C1:B(R-C1,S-C1)=C2:R=R-C1
122 IFC=ATHEN145
123 Q=C0:GOTO57
124 A(R,S-C2)=C
125 C=C+C1
126 B(R,S-C2)=C1:S=S-C1:IFC=ATHEN145
127 Q=C0:GOTO57
128 A(R+C1,S-C1)=C
129 C=C+C1:IFB(R,S-C1)=C0THEN131
130 B(R,S-C1)=C3:GOTO132
131 B(R,S-C1)=C2
132 R=R+C1
133 IFC=ATHEN145
134 GOTO89
135 IFQ=C1THEN141
136 A(R,S)=C:C=C+1:IFB(R,S-C1)=C0THEN138
137 B(R,S-C1)=C3:GOTO139
138 B(R,S-C1)=C1
```

```
139 S=S+C1:IFC=ATHEN145
140 GOTO57
141 Z=C1
142 IFB(R,S-C1)=C0THEN144
143 B(R,S-C1)=C3:Q=C0:GOTO51
144 B(R,S-C1)=C1:Q=C0:R=C1:S=C1:GOTO56
145 IFZ<>C1THENX=INT(H*RND(1))+C1:B(X,V-C1)=B(X,V-C1)+C1
146 GOSUB147:POKESC+12*LL+18-
H+LL*V+2*AA,30:POKE55296+12*LL+18-H+LL*V+2*AA,7:GOTO173
147 GOSUB189:REM PRINT *** MAZE ***
148 M1=C1:PRINT"{clear}";:IFV=11THEN150
149 FORI=C1TO11-V:PRINT:NEXT
150 FORJ=VTOC1STEP-C1:PRINTSPC(19-H)
151 FORI=C1TOH:PRINT"{reverse on} ";
152 IFB(I,J-C1)=C0THEN156
153 IFB(I,J-C1)=C2THEN156
154 PRINT"{black} {green}";
155 GOTO157
156 PRINT" ";
157 NEXTI
158 PRINT" "
159 PRINTSPC(19-H)"{reverse on} ";
160 FORI=C1TOH:PRINT"{black} {green}";
161 IFB(I,J-C1)<C2THEN164
162 PRINT"{black} {green}";
163 GOTO165
164 PRINT" ";
165 NEXTI:PRINT
166 NEXTJ
167 PRINTSPC(19-H)"{reverse on}";:FORI=C1TOH
168 IFI=AATHEN170
169 PRINT"{space*2}";:GOTO171
170 PRINT" {black} {green}";
171 NEXTI
172 PRINT" ":RETURN
173 W=AA
174 FORJ=C0TOV-C1
175 FORI=C1TOH
176 IFJTHEN179
177 IFI=WTHENA(I,J)=(C3-B(I,J))*C2:GOTO181
178 M=C1:GOTO180
179 M=(A(I,J-C1)AND2)/C2
180 A(I,J)=(C3-B(I,J))*C2+M*C8
181 IFI=C1THENM=C1:GOTO183
182 M=(A(I-C1,J)AND4)/C4
183 A(I,J)=A(I,J)+M
184 NEXTI
185 NEXTJ
186 PRINT"{clear}";
187 X=W:Y=C0:Z=C1:EL=C1:ER=C1
188 GOTO259
189 PRINT:PRINT"{clear}{down*2}
controlled using the joystick :"
190 PRINT"{down}{reverse on}u{reverse off}-
moves you one step forward,"
191 PRINT"{down}{reverse on}l{reverse off}-
turn 90 degrees to the left,"
```

```
192 PRINT"{down}{reverse on}r{reverse off}-
turn 90 degrees to the right,"
193 PRINT"{down}{reverse on}fire{reverse off}-
help!":POKE198,0:WAIT198,1
194 RETURN
195 GOSUB147:PRINT"{down}you are here {yellow}Q
{green}, direction "A$(Z)
196 POKESC+LL*12+18-H+LL*V+2*X-(LL*2)*Y,81
197 POKE55296+LL*12+18-H+LL*V+2*X-(LL*2)*Y,7
198 IFY>VTHEN474
199 PRINT"{home}{down*24}what next?";
200 JO=PEEK(56320)
201 IF(JOAND4)=0THENII=II+1:GOTO206
202 IF(JOAND8)=0THENII=II+1:GOTO208
203 IF(JOAND1)=0THENII=II+1:GOTO254
204 IF(JOAND16)=0THENII=II+5:GOTO195
205 GOTO200
206 Z=Z-C1:IFZ<C1THENZ=Z+C4
207 GOTO259
208 Z=Z+C1:IFZ>C4THENZ=Z-C4
209 GOTO259
210 RETURN
211 IFZ=C1THENA1=A-C1:B1=B-C1:GOTO216
212 IFZ=C2ANDB<VTHENA1=A:B1=B:GOTO216
213 IFZ=C3THENA1=A+C1:B1=B-C1:GOTO216
214 IFZ=C4ANDB>C1THENA1=A:B1=B-C2:GOTO216
215 EL=C1:RETURN
216 F=A(A1,B1):IFZ=C1THEN218
217 FORI=C2TOZ:F=(FAND14)/C2+(FANDC1)*C8:NEXT
218 EL=(FANDC2)/C2:RETURN
219 IFZ=C1THENA1=A+C1:B1=B-C1:GOTO224
220 IFZ=C2ANDB>C1THENA1=A:B1=B-C2:GOTO224
221 IFZ=C3THENA1=A-C1:B1=B-C1:GOTO224
222 IFZ=C4ANDB<VTHENA1=A:B1=B:GOTO224
223 ER=C1:RETURN
224 F=A(A1,B1):IFZ=C1THEN226
225 FORI=C2TOZ:F=(FAND14)/C2+(FANDC1)*C8:NEXT
226 ER=(FANDC2)/C2:RETURN
227 IFB=C0THEN235
228 IFB>VTHENE=C3:RETURN
229 F=A(A,B-C1):IFZ=C1THEN231
230 FORI=C2TOZ:F=(FAND14)/C2+(FANDC1)*C8:NEXT
231 C=FANDC1:D=(FAND4)/C4:E=(FAND2)/C2
232 IFC=C0THENGOSUB211
233 IFD=C0THENGOSUB219
234 RETURN
235 C=C0:D=C0:E=-C1
236 IFZ<>1THEN240
237 E=C1
238 IFA=WTHENE=C0
239 RETURN
240 IFZ=C3THENE=C2:RETURN
241 IFZ=C2ANDA=HTHENE=C2:RETURN
242 IFZ=C4ANDA=C1THENE=C2:RETURN
243 RETURN
244 IFE>C0THEN250
245 IFZ=C1THENB=B+C1:RETURN
```

```
246 IFZ=C2THENA=A+C1:RETURN
247 IFZ=C3THENB=B-C1:RETURN
248 IFZ=C4THENA=A-C1:RETURN
249 PRINT"{home}{down*10}{right*10}";:RETURN
250 IFE=C0THEN252
251 PRINT"{ct g}";
252 S=C1
253 RETURN
254 A=X:B=Y
255 GOSUB227
256 GOSUB244
257 X=A:Y=B
258 IFE>C0THEN198
259 A=X:B=Y
260 GOSUB227
261 REM *** SHOW 5 DEEP ***
262 FORT=C1TO5
263 GOSUB274
264 IFETHENT=C5:GOTO268
265 GOSUB244
266 GOSUB227
267 IFE>C1THENT=C5:GOTO268
268 NEXTT
269 IFX=WANDY=C0ANDZ=C1THEN273
270 IFX<>WANDY=C0ANDZ=C1THENGOSUB249:PRINT"{up*4}{right*3}
outer wall":GOTO272
271 GOTO273
272 GOSUB249:PRINT"{up*2}{right*2}you're outside !!!"
273 GOTO198
274 ONTGOTO275,311,355,399,447
275 REM *** SHOW THE DEPTH 1 ***
276 PRINT"{clear}";
277 IFE>C2THENRETURN
278 IFE<C0ORE>C1THENGOSUB251:RETURN
279 FORI=C1TO21:IFC=C1THENPRINT"{right*8}Y";:GOTO285
280 IFY=C0THENPRINT"{right}{reverse on}{red}"MID$(W$,
(IANDC1)+C1,C8)"{green}";:GOTO285
281 IFEL=C1THENPRINT"{right}{white}{cm +*8}{green}";:GOTO285
282 IFI=C2THENPRINT"{right}{cm @*7}{white}{cm +}
{green}";:GOTO285
283 IFI<18THENPRINT"{right*8}{white}{cm +}{green}";:GOTO285
284 PRINT"{right}{reverse on}{space*7}{reverse off}{white}
{cm +}{green}";
285 IFE=C0THENPRINTSPC(22);:GOTO288
286 IFY=C0THENPRINT"{reverse on}{red}"MID$(W$,(IANDC1)
+C1,22)"{green}";:GOTO288
287 PRINT"{white}{cm +*22}{green}";
288 IFD=C1THENPRINT"T":GOTO294
289 IFY=C0THENPRINT"{reverse on}{red}"MID$(W$,(IANDC1)
+C1,C8)"{green}":GOTO294
290 IFER=C1THENPRINT"{white}{cm +*8}{green}":GOTO294
291 IFI=C2THENPRINT"{white}{cm +}{green}{cm @*7}":GOTO294
292 IFI<18THENPRINT"{white}{cm +}{green}":GOTO294
293 PRINT"{white}{cm +}{reverse on}{green}{space*7}"
294 NEXT
295 FORI=C1TOC3:IFC=C1OR(X=WANDY=C0)THEN301
296 IFI<>C1THEN298
```

```
297 IFEL=C0THENPRINT"{right}{reverse on}{cm t*7}N";:GOTO302
298 PRINT"{right}{reverse on}"LEFT$(SP$,C8-I)"N"LEFT$(SP$,I-
C1);
299 IFI=C1THEN302
300 GOTO303
301 PRINTSPC(C9-I)"{reverse on}{sh pound}"LEFT$(SP$,I-
C1);:IFI<>C1THEN303
302 IFE=C0THENPRINT"{reverse on}{cm t*22}";:GOTO304
303 PRINT"{reverse on}{space*22}";
304 IFD=C1OR(X=WANDY=C0)THEN308
305 IFI<>C1THEN307
306 IFER=C0THENPRINT"{reverse on}M{cm t*7}":GOTO309
307 PRINT"{reverse on}"LEFT$(SP$,I-C1)"M"LEFT$(SP$,C8-
I):GOTO309
308 PRINT"{reverse on}"LEFT$(SP$,I-C1)"{cm asterisk}"
309 NEXTI
310 RETURN
311 REM *** SHOW THE DEPTH 2 ***
312 PRINT"{home}";:Z$="{right*9}"
313 FORI=C1TOC2:PRINTZ$;:IFC=C0THEN316
314 PRINTSPC(I+C1)"M";SPC(C2-I);:GOTO319
315 PRINT"{right*3}M";:GOTO319
316 IFI=C1THENPRINTSPC(C4):GOTO319
317 IFEL=C1THENPRINT"{cm @*4}";:GOTO319
318 PRINT"{right*3}{cm @}";
319 IFE=C0ORI=C1THENPRINTSPC(14);:GOTO321
320 PRINT"{cm @*14}";
321 IFD=C0THEN323
322 PRINTSPC(C2-I)"N":GOTO326
323 IFI=C1THENPRINT:GOTO326
324 IFER=C1THENPRINT"{cm @*4}":GOTO326
325 PRINT"{cm @}"
326 NEXT
327 FORI=C1TO15:PRINTZ$;:IFC=C1THENPRINT"{right*3}
Y";:GOTO332
328 IFEL=C1THENPRINT"{reverse on}{white}{cm +*4}
{reverse off}{green}";:GOTO332
329 IFI=C3THENPRINT"{cm @*3}{reverse on}{white}{cm +}
{reverse off}{green}";:GOTO332
330 IFI>12THENPRINT"{reverse on}{space*3}{white}{cm +}
{reverse off}{green}";:GOTO332
331 PRINT"{space*3}{reverse on}{white}{cm +}{reverse off}
{green}";
332 IFE=C0THENPRINT"{right*14}";:GOTO334
333 PRINT"{reverse on}{white}{cm +*14}{reverse off}{green}";
334 IFD=C1THENPRINT"T":GOTO339
335 IFER=C1THENPRINT"{reverse on}{white}{cm +*4}
{green}":GOTO339
336 IFI=C3THENPRINT"{reverse on}{white}{cm +}{reverse off}
{green}{cm @*3}":GOTO339
337 IFI>12THENPRINT"{reverse on}{white}{cm +}{green}
{space*3}":GOTO339
338 PRINT"{reverse on}{white}{cm +}{green}"
339 NEXT
340 FORI=C1TO4:PRINTZ$;:IFC=C1THEN345
341 IFI<>C1THEN344
342 IFEL=C0THENPRINT"{reverse on}{cm t*3}N";:GOTO346
```

```
343 PRINT"{reverse on}{space*3}N";:GOTO346
344 PRINT"{reverse on}"LEFT$(SP$,C4-I)"N"LEFT$(SP$,I-
C1);:GOTO347
345 PRINTSPC(C4-I)"{reverse on}{sh pound}"LEFT$(SP$,I-
C1);:IFI<>C1THEN347
346 IFE=C0THENPRINT"{reverse on}{cm t*14}";:GOTO348
347 PRINT"{reverse on}{space*14}";
348 IFD=C1THEN352
349 IFI<>C1THEN351
350 IFER=C0THENPRINT"{reverse on}M{cm t*3}":GOTO353
351 PRINT"{reverse on}"LEFT$(SP$,I-C1)"M"LEFT$(SP$,C4-
I):GOTO353
352 PRINT"{reverse on}"LEFT$(SP$,I-
C1)"{cm asterisk}":GOTO353
353 NEXTI
354 RETURN
355 REM *** SHOW THE DEPTH 3 ***
356 PRINT"{home}{down*2}";:Z$=Z$+"{right*4}"
357 FORI=C1TO3:PRINTZ$;:IFC=C0THEN359
358 PRINTSPC(I-C1)"M"SPC(C3-I);:GOTO362
359 IFI<C3THENPRINT"{right*3}";:GOTO362
360 IFEL=C1THENPRINT"{cm @*3}";:GOTO362
361 PRINT"{right*2}{cm @}";
362 IFE=C0ORI<=C2THENPRINT"{right*8}";:GOTO364
363 PRINT"{cm @*8}";
364 IFD=C0THEN366
365 PRINTSPC(3-I)"N":GOTO369
366 IFI<C3THENPRINT:GOTO369
367 IFER=C1THENPRINT"{cm @*3}":GOTO369
368 PRINT"{cm @}"
369 NEXT
370 FORI=C1TO9:PRINTZ$;:IFC=C1THENPRINT"{right*2}Y";:GOTO375
371 IFEL=C1THENPRINT"{white}{cm +*3}{green}";:GOTO375
372 IFI=C2THENPRINT"{cm @*2}{white}{cm +}{green}";:GOTO375
373 IFI>C7THENPRINT"{reverse on}{space*2}{reverse off}
{white}{cm +}{green}";:GOTO375
374 PRINT"{space*2}{white}{cm +}{green}";
375 IFE=C0THENPRINT"{right*8}";:GOTO377
376 PRINT"{white}{cm +*8}{green}";
377 IFD=C1THENPRINT"T":GOTO382
378 IFER=C1THENPRINT"{white}{cm +*3}{green}":GOTO382
379 IFI=C2THENPRINT"{white}{cm +}{green}{cm @*2}":GOTO382
380 IFI>C7THENPRINT"{white}{cm +}{green}{reverse on}
{space*2}{reverse off}":GOTO382
381 PRINT"{white}{cm +}{green}"
382 NEXT
383 FORI=C1TO3:PRINTZ$;:IFC=C1THEN389
384 IFI<>C1THEN386
385 IFEL=C0THENPRINT"{reverse on}{cm t*2}N";:GOTO390
386 PRINT"{reverse on}"LEFT$(SP$,C3-I)"N"LEFT$(SP$,I-C1);
387 IFI=C1THEN390
388 GOTO391
389 PRINTSPC(C3-I)"{reverse on}{sh pound}"LEFT$(SP$,I-
C1);:GOTO387
390 IFE=C0THENPRINT"{reverse on}{cm t*8}";:GOTO392
391 PRINT"{reverse on}{space*8}";
392 IFD=C1THEN396
```

```
393 IFI<>C1THEN395
394 IFER=C0THENPRINT"{reverse on}M{cm t*2}":GOTO397
395 PRINT"{reverse on}"LEFT$(SP$,I-C1)"{reverse on}
M"LEFT$(SP$,C3-I):GOTO397
396 PRINT"{reverse on}"LEFT$(SP$,I-C1)"{cm asterisk}"
397 NEXTI
398 RETURN
399 REM **** SHOW THE DEPTH 4 ***
400 PRINT"{home}{down*5}";:Z$=Z$+"{right*3}"
401 FORI=C1TO2:PRINTZ$;:IFC=C0THEN403
402 PRINTSPC(I-C1)"M"SPC(C2-I);:GOTO406
403 IFI=C1THENPRINT"{right*2}";:GOTO406
404 IFEL=C1THENPRINT"{cm @*2}";:GOTO406
405 PRINT"{right}{cm @}";
406 IFE=C0ORI=C1THENPRINT"{right*4}";:GOTO408
407 PRINT"{cm @*4}";
408 IFD=C0THEN411
409 IFI=C1THENPRINT"{right}";
410 PRINT"N":GOTO414
411 IFI=C1THENPRINT:GOTO414
412 IFER=C1THENPRINT"{cm @*2}":GOTO414
413 PRINT"{cm @}"
414 NEXT
415 FORI=C1TO5:PRINTZ$;:IFC=C1THENPRINT"{right}Y";:GOTO420
416 IFEL=C1THENPRINT"{reverse on}{white}{cm +*2}
{reverse off}{green}";:GOTO420
417 IFI=C1THENPRINT" {reverse on}{white}{cm +}{reverse off}
{green}";:GOTO420
418 IFI>4THENPRINT"{reverse on}{cm t}{white}{cm +}
{reverse off}{green}";:GOTO420
419 PRINT" {reverse on}{white}{cm +}{reverse off}{green}";
420 IFE=C0THENPRINT"{right*4}";:GOTO422
421 PRINT"{reverse on}{white}{cm +*4}{reverse off}{green}";
422 IFD=C1THENPRINT"T":GOTO427
423 IFER=C1THENPRINT"{reverse on}{white}{cm +*2}
{green}":GOTO427
424 IFI=C1THENPRINT"{reverse on}{white}{cm +}{reverse off}
{green}{cm @}":GOTO427
425 IFI>C4THENPRINT"{reverse on}{white}{cm +}
{green} ":GOTO427
426 PRINT"{reverse on}{white}{cm +}{green}"
427 NEXT
428 FORI=C1TO2:PRINTZ$;:IFC=C1THEN434
429 IFI<>C1THEN431
430 IFEL=C0THENPRINT"{reverse on}{cm t}N";:GOTO437
431 PRINT"{reverse on}"LEFT$(SP$,C2-I)"N"LEFT$(SP$,I-C1);
432 IFI=C1THEN437
433 GOTO438
434 IFI=C1THENPRINT"{right}";
435 PRINT"{reverse on}{sh pound}";
436 IFI=C2THENPRINT" ";:GOTO438
437 IFE=C0THENPRINT"{reverse on}{cm t*4}";:GOTO439
438 PRINT"{reverse on}{space*4}";
439 IFD=C1THEN443
440 IFI<>C1THEN442
441 IFER=C0THENPRINT"{reverse on}M{cm t}":GOTO445
442 PRINT"{reverse on}"LEFT$(SP$,I-C1)"M"LEFT$(SP$,C2-
```

```
I):GOTO445
443 PRINT"{reverse on}";:IFI=C2THENPRINT" ";
444 PRINT"{cm asterisk}"
445 NEXTI
446 RETURN
447 REM *** SHOW THE DEPTH 5 ***
448 PRINT"{home}{down*7}";:Z$=Z$+"{right*2}"
449 PRINTZ$;:IFC=C0THEN451
450 PRINT"M";:GOTO452
451 PRINT"{cm @}";
452 IFE=C0THENPRINT"{right*2}";:GOTO454
453 PRINT"{cm @*2}";
454 IFD=C0THEN456
455 PRINT"N":GOTO457
456 PRINT"{cm @}"
457 FORI=C1TO3:PRINTZ$;:IFC=C1THEN459
458 PRINT"{white}{cm +}{green}";:GOTO460
459 PRINT"Y";
460 IFE=C0THENPRINT"{right*2}";:GOTO462
461 PRINT"{white}{cm +*2}{green}";
462 IFD=C0THENPRINT"{white}{cm +}{green}":GOTO464
463 PRINT"T"
464 NEXT
465 PRINTZ$;:IFC=C1THEN467
466 PRINT"{reverse on}N";:GOTO468
467 PRINT"{reverse on}{sh pound}";
468 IFE=C1THENPRINT"{reverse on}{space*2}";:GOTO470
469 PRINT"{reverse on}{cm t*2}";
470 IFD=C1THEN472
471 PRINT"{reverse on}M":GOTO473
472 PRINT"{reverse on}{cm asterisk}"
473 RETURN
474 PRINT"{clear}{down*2}you're outside !"
475 IFM1THENPRINT"{down*4}":GOTO478
476 PRINT"{clear}here is the maze":GOSUB211
477 GOSUB147
478 PRINT"you've got "II" steps used":INPUT"{down*4}
another game";Z$
479 IFLEFT$(Z$,1)="y"THENCLR:GOTO33
```