GFA Compiler

This fast 2-pass compiler converts your finished GFA BASIC programs into compact, faster-running machine language files, that operate without the aid of a runtime module.

for the Atari ST

GFA BASIC
Compiler

for the Atari ST Series

Written by GFA Systemtechnik

Distributed by
MichTron Inc.
576 South Telegraph
Pontiac, MI 48053
Tel: (313)-334-5700
BBS: (313)-332-5452

TABLE OF CONTENTS

Introduction

There are two kinds of programming languages: interpreter languages and compiler languages. Characteristic of an interpreter is fast development and testing of (short) programs. Compiler languages - like the teaching language Pascal - require more effort in the coding of the program, and tests are as a rule difficult to carry out. In return the programs are - usually - faster and are automatically protected against listing of the source code.

The GFA BASIC Compiler converts programs written with the GFA BASIC Interpreter into machine-language programs. That means simply that your programs will run faster and will require neither GFA BASIC nor its associated RUN ONLY interpreter.

It will compile only those programs that have been stored from the GFA BASIC Interpreter in the SAVE mode; not with SAVE A, LIST or PSAVE.

The compiler requires no extra linker or even assembler, but produces directly a complete machine-language program.

Operation

1. Write a program with the interpreter and test it as usual.

2. Save the program with SAVE on a diskette and leave the interpreter.

3. Insert the compiler diskette and start the compiler with a double-click.

4. The video screen shows an OPTIONS window, through which you can control compiler operation.

5. After you select the COMPILE box or press the <RETURN> key, a FILESELECT box appears, from which you may choose the program to be compiled.

6. The compiler reads the program and checks the program structure (DO loops and so forth), operating as a counterpart to the interpreter.

7. The compiler reads the program and translates it into machine code.

8. The compiler follows with a FILESELECT box for selecting the name under which the compiled program will be known.

9. Then the OPTIONS window reappears (Step 4). You may compile other programs or return to the GEM desktop with ABORT.

10. Start your program with a double-click.

Options

In GFA BASIC (after Version 2.0) a command OPTION "text" is provided, which is useful for extra control of the compiler. In addition, you can make adjustments with the window displayed at the beginning of the compiler .

You can make the following adjustments:

OPTION "U?" **STOP ???** on dialog box

Adjustment of the response to interruptions through the three STOP keys:

<SHIFT>+<ALTERNATE>+<CONTROL>.

OPTION "T?" **TRAPV ???** on dialog box

Does the program always react to overflow in integer arithmetic?

OPTION "E?" **ERRORS ???** on dialog box

Option to show error messages as text in the compiled program, or to save disk space by not including the error messages in the compilation.

OPTION "B?" **BOMBS ???** on dialog box

Will ignore or accept BOMB error messages (those with error numbers greater than 100).

Interruption

OPTION "Ux" or **STOP xxxx**

With this option, you can choose whether and when the STOP key:

<SHIFT>+<ALTERNATE>+<CONTROL>

will be answered. The more often the query is made, the longer and slower the program becomes (each query costs four bytes of memory space).

OPTION "U0" **STOP NEVER** on dialog box

Shuts off the Stop key query.

OPTION "U1"

Inserts **exactly one** query at this point.

OPTION "U2" **STOP LOOP** on dialog box

After this, a query will occur **before** the following commands: GOTO, LOOP, UNTIL, WEND and NEXT.

OPTION "U3" **STOP EVER** on dialog box

After this, a query will be made **after** each command.

An example of OPTION "Ux":

```
OPTION "U0"                    !Disable Stop
PRINT "Stop not on"
REPEAT
    PRINT "*";
UNTIL MOUSEK AND 1             !until left Mouse key
'
ON BREAK GOSUB HELP
DO                             !Endless loop
    OPTION "U1"
    PRINT "#";
LOOP
'
PROCEDURE HELP                 !Exit with Sh-Alt-Ctrl
    PRINT "Help";
    IF MOUSEK AND 2            !and right Mouse key
        END
    ENDIF
RETURN
```

This small program first runs in a loop; printing stars. It cannot then be halted by pressing the STOP keys <SHIFT>+<ALTERNATE>+<CONTROL>. After you press the left mouse button, the program runs in the next loop, printing crosses. Now the program can be interrupted with the three keys. The subprogram is now called (ON BREAK GOSUB), from which you can end the program by pressing the right mouse button.

Important: In the procedure for handling the STOP key, you should choose OPTION "U0,". Insert OPTION "U0" directly before the procedure and immediately after the RETURN command; turn the interrupt on again.

Integer overflow

In compiled programs, a few arithmetic commands (concerning integers) will be directly carried out by the program without a call to any subprogram (for example, INC A%). Most of the time one can proceed without using a test of overflow; since this costs time and space and since for a range of -2 billion to + 2 billion, overflow hardly ever occurs.

OPTION "T+" **TRAPV** + on dialog box

After this command, the arithmetic commands that follow will be combined with a TRAPV command. When an error is found a display seven bombs or error message #107 will occur.

OPTION "T-" **TRAPV** - on dialog box

After this command the call to TRAPV is eliminated.

Error text

For errors in compiled programs, a short error message is usually displayed, for example:

Error # -033, PC>$00xxxxxx.

Through an option, it is possible to include, in the compiled program, a somewhat longer error message. "Error -33" then appears as "Data not found."

OPTION "E+" **ERRORS TEXT** on dialog box

Error messages appear as text when this command is used. This uses some memory space, but makes the detection of errors much simpler.

OPTION "E-" **ERRORS NO.** on dialog box

Here no error messages are included. Therefore any error messages appear only as numbers on the video screen.

The last setting applies that when OPTION "E-" stands as the last command in a program, no error messages are included, independent of the preceding settings.

With the error messages there still remains one possibly confusing statement:

PC>$00xxxxxx.

This shows the position of the program counter of the 68000 CPU at the last test of the STOP key (OPTION "U?").

Bombs

The Atari is known for frequently handing out bombs. The bombs are shown on the screen because an exception has been encountered that is not defined; for example, locating the beginning of a word on an uneven storage address. The GFA BASIC Interpreter catches most of these errors and then often (though not always) is ready to work and the program and data are still intact. One can likewise choose to apply these catches of bomb errors in compiled programs.

OPTION "B+" **BOMBS** + on dialog box

OPTION "B-" **BOMBS** - on dialog box

When "B+" or BOMBS + has been chosen last, the appropriate catch routines are inserted into the compiled program.

Then, for example, a DPEEK(INTIN+3) will not lead to 3 bombs and loss of program and data, but only to ERROR 103, which can be caught with ON ERROR GOSUB xxx and RESUME.

Command differences

There are a very few commands that have meaning only for the interpreter and cannot be carried out by the compiler:

LIST	LLIST
TRON	TROFF
DEFLIST	SAVE
LOAD	PSAVE
STOP	CONT

These commands require either a listing of the program or are meaningless in compiled programs (PSAVE), and also are not especially useful in interpreter programs.

CHAIN "filename"

Interpreter:

First, the filename attached will be examined to determine whether there is a "." after the last "\" (that is, whether the filename has an extension). If not, ".BAS" will be appended to the filename. Then the file will be opened, checked to see if it is a GFA BASIC program (interpreter), then loaded and started. For erroneous filenames, an error message will be displayed or else the ON-ERROR-GOSUB routine called.

Compiler:

The filename will in no case be altered. The given filename will be given over to the GEM-AES function sh_write and the compiled program ended with QUIT. On return to the GEM Desktop, the corresponding program is called, exactly as if with a double-click and with the usual white title line with the program name on the screened (colored) Desktop background. The program called occupies no more storage area after this command. Together with the program, sh_write also will write the contents of BASEPAGE +128 as a command line.

That sounds complicated, as it is; nevertheless, a few examples should clear things up:

```
CHAIN "DATEI2.PRG"
```

Here the program (compiled by GFA BASIC) is loaded from the diskette and started.

```
A$="TEST.DOC"
A$=CHR$(LEN(A$))+A$+CHR$(0)
BMOVE VARPTR(A$),BASEPAGE+128,128
CHAIN "1ST_WORD.PRG"
```

Here 1ST_WORD (a word processing program by GST, distributed by Atari) is loaded and started. The text file "TEXT.DOC" is preselected, so that 1ST_WORD shows no FILESELECT box. The two complementary following commands write the filename under the GEMDOS convention on the upper half of the base page. Through the CHAIN command this command line is then transferred to GEM. GEM then writes this line in the appropriate spot in the program called.

```
POKE BASEPAGE+128,0
CHAIN "1ST_WORD.PRG"
```

Here 1ST_WORD is also loaded, but it is guaranteed to be without the document name. Without the POKE command, it could happen that a name placed in the basepage could be encountered.

See also: EXEC (Interpreter manual V 2.0, Appendix F14)

FILESELECT "path", "file", file$

In compiled programs there must be at least 32,500 bytes free before calling this command, because the compiler reserves no fixed secondary storage. The interpreter reserves a few fixed storage areas of 32 Kbytes for an editor and can then also use these for the FILESELECT call in order to save the old screen background. The compiler on the contrary uses no fixed background storage, but instead uses dynamic string regions.

RESUME
RESUME NEXT

The RESUME command without declaration of a label is little different than it is under the interpreter. RESUME or RESUME NEXT must know, respectively, where the corresponding command begins or ends. To that end a large table would have to be attached to the program that retains the distance to each compiled command.

Instead, the GFA BASIC Compiler stores only the interval between the appearances of OPTION "Ux." The program counter necessary for the resumption of the program will also be saved there, so that a RESUME is (usually) possible after TOS errors as well. The "side effect" that OPTION "Ux" also lets the computer check the three STOP keys permits one to break off through ON BREAK CONT or, even more usefully, use ON BREAK GOSUB xxx to reach a defined break routine.

The example program on the next page seeks first to load the data "MCODE.DAT" into the string A$. If the data cannot be loaded - for whatever reason - an error message is displayed. Although the program runs in a loop, which theoretically can be ended only through a RETURN, one can stop the program also with the three STOP keys after a certain question.

Example of ON ERROR/RESUME/ON BREAK and OPTION "Ux"

```
,
,
OPTION "U0"
ON BREAK GOSUB STOP
ON ERROR GOSUB ERROR
OPTION "U1"
OPEN "I",#1,"MCODE.DAT"
A$=INPUT$(LOF(#1),#1)
CLOSE #1
ON ERROR
,
,
PROCEDURE ERROR
    CLOSE #1
    PRINT "Cannot load" "MCODE.DAT"
    PRINT "Error number" 'ERR
    PRINT "Please fix error"'
    PRINT "and press RETURN"
    REPEAT
        OPTION "U1"
    UNTIL INKEY$=CHR$(13)
    RESUME
RETURN
PROCEDURE STOP
    PRINT "Program stop (Y/N)?";
    REPEAT
        A$=UPPER$(INKEY$)
        IF A$="J$"
            END
        ENDIF
    UNTIL A$="N"
    PRINT
RETURN
```

Surprises

What is the result of the following program:

```
FOR I=-2 TO 2
    PRINT I;"/";I"=";I/I
NEXT I
```

Entirely clear!!!?!

Naturally:	-2/-2=1	
	-1/-1=0	
	0/0=	Division by Zero
But what do you think!	-2/-2=1	
	-1/-1=1	
	0/0=1	What next??
	1/1=1	
	2/2=1	

Why????

The compiler is fairly dumb, but it notices when something is to be divided by itself and writes a 1, using the program:

```
PRINT I;"/";I;"=",1
```

The case that one divides 0 by 0 is rare (and foolish) enough, that one safely can substitute the value for the division. The same goes also for addition, subtraction and multiplication; thus (A+B)*(A+B) is converted to the much faster double(A+B).

In general one may not expect too much of this optimization: A+B-B will not be recognized, though B-B+A will.

Speed

There is a difference between the running speed of an interpreted program and that of the corresponding compiled program (hopefully). There are, however, commands that are more accelerated than others by the compiler. So especially fast compiled programs often come from especially slow interpreter programs.

The principal advantage of the compiler is the better use of integer arithmetic. For example, for INC A% a single machine instruction is required, though for the interpreter there must always be some 50 commands, of which 49 serve only to determine the operation to be carried out and have nothing to do with the particular program. On the other hand, the corresponding interpreter commands play hardly any role in a sine calculation because it hardly matters whether 5000 or 5050 commands are carried out. (5,000 steps to do the numerical approximation)

Especially revealing is a comparison of integer variables. Thus it happens that the fastest numerical loop with integer variables is not, as for the interpreter, the FOR-NEXT loop. Indeed, this loop is faster under the compiler than it is under the interpreter, but its speed is surpassed by the REPEAT-UNTIL loop under the compiler.

```
FOR I%=1 TO 1000
NEXT I%
```

is therefore slower than:

```
I%=0
REPEAT
    INC I%
UNTIL I%>1000
```

Compatibility

If you want to write a program for which it is important how much storage space a number occupies because for example, you want to save a number field on the diskette with BSAVE, please calculate in your program:

```
bytes%=VARPTR(a(1))-VARPTR(a(0))
```

Then you can transfer your BASIC program, should the number format should be altered sometime.

License fees

The programs constructed with the GFA BASIC Compiler may be distributed without payment of a license fee.

The name of the programming language, GFA BASIC, however, should be included on the diskette, packaging or instructions. (Thanks)

Internals

The calls for the routines in the runtime library (mathematical routines for the most part) arise through jsr xxx(a6).

Especially for OPTION "U" is jsr (a6) used (only by the compiler).

There is a fixed address for interpreted and compiled programs, at which also machine programs can begin to run, in order to respond to errors. It is possible to call the normal BASIC error routines through

```
moveq #num,d0
jmp 2(a6)
```

Before CALL and C: all registers are automatically saved (a3-a6). It is not necessary to save other registers in a machine routine.

YOUR RIGHTS AND OURS: This copy of ***GFA BASIC COMPILER*** is licensed to you. You may make copies for your own use or for archival storage. You may also sell your copy without notifying us. However, we retain copyright and other property rights in the program code and documentation. We ask that ***GFA BASIC COMPILER*** be used either by a single user on one or more computers or on a single computer by one or more users. If you expect several users of ***GFA BASIC COMPILER*** on several computers, contact us for quantity discounts and site-licensing agreements. Also if you intend to rent this program, or place this program on a BBS, contact us for the appropiate license and fee.

We think this user policy is fair to both you and us; please abide by it. We will not tolerate use or distribution of all or part of **GFA BASIC COMPILER** or its documentation by any other means.

LIMITED WARRANTY: In return for your understanding of our legal rights, we guarantee **GFA BASIC COMPILER** will reliably perform as detailed in this documentation, subject to limitations here described, for a period of thirty days. If **GFA BASIC COMPILER** fails to perform as specified, we will either correct the flaw(s) within 15 working days of notification or let you return **GFA BASIC COMPILER** to the retailer for a full refund of your purchase price. If your retailer does not cooperate, return **GFA BASIC COMPILER** to us. While we can't offer you more cash than we received for the program, we can give you this choice: 1) you may have a cash refund of the wholesale price, or 2) you may have a merchandise credit for the retail price, which you may apply toward buying any of our other software. Naturally, we insist that any copy returned for refund include proof of the date and price of purchase, the original program disk, all packaging and documentation, and be in salable condition.

If the disk on which **GFA BASIC COMPILER** is distributed becomes defective within the warranty period, return it to us for a free replacement. After the warranty period, we will replace any defective program disk for $5.00.

We cannot be responsible for any damage to your equipment, reputation, profit-making ability or mental or physical condition caused by the use (or misuse) of our program.

We cannot guarantee that this program will work with hardware or software not generally available when this program was released, or with special or custom modifications of hardware or software, or with versions of accompanying or required hardware or software other than those specified in the documentation.

Under no circumstances will we be liable for an amount greater than your purchase price.

Please note: Some states do not allow limitations on how long an implied or express warranty lasts, or the exclusion or limitation of incidental or consequential damages, so some of the above limitations or exclusions may not apply to you.

UPGRADES AND REVISIONS: If you return your information card, we will notify you if upgrades to **GFA BASIC COMPILER** become available. For minor upgrades and fixes, return the original disks to us with $5.00. For major revisions, the upgrade fee is typically 15-20% of the original suggested retail price.

FEEDBACK: Customer comments are VERY important to us. We think that the use, warranty and upgrade policies outlined above are among the fairest around. Please let us know how you feel about them.

Many of the program and documentation modifications we make result from customer suggestions. Please tell us how you feel about **GFA BASIC COMPILER** - your ideas could make the next version better for all of us.

COPYRIGHT NOTICE:

GFA Compiler

by GFA Systemtechnik

576 S. Telegraph, Pontiac, MI 48053
Orders and Information (313) 334-5700

GFA-Basic Compiler

Atari ST Fast 2-pass Compiler
by GFA

Nº 3912