# BASTOOL

A BASIC PORTING SWISSKNIFE TOOL

# BETA

**DONT USE IT FOR PRODUCTION WORK**
**THE PROGRAM IS DELIVERED WITHOUT WARRENTY**
**USE IT ON YOUR OWN RISK**
**DONT REDISTRIBUTE WITHOUT THIS TEXT**
**ALL RIGHTS RESERVED**

## MAKE A BACKUP OF THE CODE YOU WILL PROCESS FIRST !

# Table of Content

# Hello and welcome to bastool

bastool ins mainigly intend to help you port old or stonehenge old basic programs to free basic or to a other more modern basic dialect. Usually you has only to change a few lines in the old code
most of them will be  errors...

## *What it does:*

It works on correct clean GW-BASIC styled source. It currently makes some nice syntax highlighting, trys to detect variable definitions and makes some code indentations. (if you whish)

It renames numeric labels to non numric ones (if you whish)

It detects used and unused Labels,clean up the code (just a bit), split up lines (if you whish)

It insert „end if“ statements (if you whish)

It gives braces a colorful meaning of the owner, trys to repair broken string, and broken print statements (if you whish)

It trys to rewrite „def fn“ statemenst on a primitive manner
(you has too look at the new code !!! )

It will renumber your code also if line numbers are not always be present. (if you whish)

It makes cross reference tables for variables, jump labels, used reserved words (if you whish)

It will make a list with the used strings so you can simply do a localisation of your program (if you whish).

## *What it NOT does:*

Bastool is not a press a button, i will suggest you – that i do it all, tool.

It will not handles big files
(you have to redefine the dim statements if you has a need for)

It will not handles all crippled code (SHARP Basic), even if i work on some of these things.

It will not make perfect results on every „stonenge old“ Basic

I doesnt cares about all forbidden or obsolete Words – you can enhanced it by modifying the word files.
(it always try to do its best but...)

It will no redesign the whole program after consulting your marketing experts, optimise and pollish it or correct failures.

It works not on all freebasic source because multiline comments are not handled correctly (*If bastools code looks like a real beauty, then i will have done this ;)*

And of course it will not allways work completely correct ;)

## *Planned features:*

detokenize old gw basic code  (most parts are prepared for that right now)
full commandline interface for all possible settings
work on any or more basic... ()
reduce memory consumption
code cleanup
printing output
code linebreak
code to html
handling libs
write out and reinsert vars and strings
read kate syntax files
write kate syntax files

# INSTALLATION

Simply put the source in a subdirectory.
If you are working under dos you might have to use the distributed „carriage linefeed“ tools to put the needed CR+LF in the code.

Bastool also needs a subdirectory which contains the reserved word files (its hardcodet at the moment)

Compile it now with the freebasic compiler

```
fbc -exx bastool.bas
```

Run it, Help will given on run.

## *A note to the words files*

Each line contains a reserved word without spaces in front or end,
they might be unsorted or contain double entrys
(bastool will optimize them internally)

if you want to help me for the next version send me your words files it would be great if you mark functions with a "(" at  the end of the word and put a type as text word.  So it could look like:

```
left$( string
```

thx in advance

## *Bug reports*

They are wellcome. but please be kind and fix the code first in the original source. (It is not made for doing error corrections in the source). It should be able to run in gw or qbasic at least. If you report a bug please ship the bug with the file you would like to have converted. And be a bit patient pease. I do this as a spare time project, i doesnt get any money for this ;)

### *Algorithm*

Very simple ;) it looks for the reserved words of the originally used basic interpreter / compiler. All things which are not a special char or a reserved word must be a variable identifyer or  a label.

Then it does some magic with renumbering etc. and care for the words which are used by the TARGET interpreter or compiler to avoid conflicts

## HOW TO WORK WITH

### *Old code:*

In the early years the computer memory was expensive and restricted. My first computer has had 7802 Basic bytes free (or so) for the code and and the datas. So the goal for the basic developers at that times was to save memory. So strings where not closed at the end, space signs was left out and more just to get some extra bytes. Another commob pitfall are jumpmarks which not match a real existing line number.
(The code has worked cause the interpreter used the next line bevore the missed mark)
I suggest to fix this in the original source first. Bastool will help to find those miracles and its always a good thing to have valid code.

### *Labels:*

The renamed labels are ready for simple renaming later (by hand) to a more meaningful name. Thats why they contain some extra chars. A Label „10“ would be otherwise replaced in a „for next“ loop...

Unused labels can also be simple removed later by a single replace with a regular expression. (you can also drop those labels). Sometimes it is useful to let those labels in your code. For example if your code jumps to not existing labels The labels are named in a manner that you can see in a normal editor how they are used in the code.

Used marks starts with L_ (Goto, Then etc. Jump)

If they are a jump mark from a „gosub“ then they starts with SUB_ this will override the „goto“ label mark, it should give you a hint to make a „sub“ statement from such a block.

Normal labels will start with "l" to make them different from normal numbers and labels which starts with "n" indicate that they are created by a bastool line split.

### *Ident:*

Often old code contains never fixed bugs. So if the indent doesnt go back to the starting (zero) something in the code usually has to be reworked.
it will not fix this ! (would you give some extra lessions in debugging)
For example: In freebasic it's forbiden to do such things like
if i=67 then next.
The indent will give you a hint where this is broken.
The error is always located bevore the hint !

### *EndIf:*

This switch is ment for old code which only contains single line „if“ statements, the switch is useful if you want to break up the code by the ":" chars.
If you are for example on turbo basic or powerbasic which already contains such stetments turn this off. Otherwise you will get some "endifs" for free ;)

### *Var:*

Bastool looks for the USED vars by „peek“, „input“, „read“, the „equal sign“ etc. So if a variable never got a value, then this variable will not noticed as a variable and not be renamed. If that happens you has found a bug in the old code usually. It will need deeper investigation, thats why bastool will not fix it !

Your compiler will show you this disaster with a nice line numbers. A short remark - its much harder to debug a potentielly working source then a real broken ones...

Herman has ask why i do such a variable renaming. Well its simple to understand. My goal is to port old code to a modern basic. Freebasic doesnt allow vars with the type chars (!$%#) at the end if you end the qbasic compatibility mode.

So there will be no hint in the code of which type a var was made in the beginning. All vars in freebasic mode also have to be declared explicit. Vars with the same name as Array names are forbidden. (A$ will produce a error with A$(n) ). Usually you have to rename nearly all of them and also to produce the „Dim“ statements. Its hard to make the renaming by hand without getting errors in the code. (Especially if you are not the author of the source.)
I am here dealing with some engineering source. A failure there will break this source, and a 3000 lines stress calculatio program isnt as easy to debug as "hunt the wumpus" ...

### *Reserved Words:*

You will see that bastool arrange spaces around every known reserved word (wordfile content) or symbol (symbolfile content)

Thats made for two reasons.
First it makes the code more easy to read. (Well not as urgent for some ;) )
But second, you are now able to replace Reserved words by
a simple search and replace. Maybee if the old basic requires
ARCTAN and you has to use the freebasic ATAN function.
So search simply for " ARCTAN " with spaces and replace them

### *Braces:*

Bastool will replace every "(" with " ( " same as bevore with the reserved words.
Some Interpreters or Compilers will fail with that cause the "(" has to be glued directly to the function or var. Just do a replacement in your favorite editor
Search for " (" and replace it with "(" and you are done :)
Bastool also shows you braces in different colors (in the console output)
Mathematical needed ones will be shown as white ones logical ones a red ones,
Array ones in magenta and those needed by a reserved word will appear in blue and so on...

This is made for curious engineers who has the whish to see a mathematical rule in the code. (its also useful for a later revision of bastool ...)

### *General:*

In fact im happy if i can fix things in the old source first. There is a lot of old code laying in the net which has bugs, and these bugs are sittin in the code since (over) 20 ? or more years. Never found by lazy interpreters or programmers (fat grin).

**I can not stress this enough - its better to fix original source bugs first !**

## Hints

You can rediret the output to a file, the console color is only shown on he console so you could gives the dumped file directly a try with freebasic. On this way bastool should be also able to get invoked by a marked region in kate...

Most of the Reserved Words can be changed in the words.txt file
but things like „IF“, „THEN“, „FOR“, „NEXT“, „DO“, „LOOP“ ect. are hardcoded right now

## Code

As you will see in the code, this is work progress. Usually it will do most of the job but currently not always perfect. And some parts are a bit strange at this stage :-“ Im finish if bastool can clean up its own code !

(BTW: usability is a unknown word at this point even if i put work on it)
it is currently also only tested in a LINUX Environment
dont cry for a dos or Windows version you can compile them byself. Bastool works OS independing (Maybee you has to replace the few „/“ chars in the file loading section by „\“ ones and also to put CR+LF in the code, but toold for that are shipped with bastool)
I also never trys bastool in dos - for my opinion are over 20 years of 64k limits enough !

Comments, suggestions, gifts, postcards,
beer and MONEY *rofl* (whats that ?) are always welcome ;)

## Credits

Thanks to Herman Wigman for help on debugging Bas2txt and some hints and comments and giving me the code for gw2qbas which wasnt used but a help

you can run both tools together. Herman will eliminate

a lot of ugly code (goto). You can use Bastool bevore and after but dont use var replacing and label replacing bevore you use hermans gw2qbas.

Bastool will be a help, if you got a mix of gwbasic and qbasic gw2bas needs urgend line numbers at the moment and bastool will make them. After gw2bas you can do all the optimizing things for freebasic

Ebben Feagan for fixing Freebasic to make the code from Herman compile.

Dipl.-Ing. Bernhard Falter, Dipl.-Ing. Martin Dietz who wrote a stress calculating system in GW – Qbasic which gaves the intention to do this work.

All the Peoples around the FreeBasic Compiler, and all the peoples who does the other Compiler and Interpreters ive had the joy to play with :)

And last but not least all the brave who host the old source to save
some social and cultural history. It was fun to make some games work.
(:-")

# Some Words about Basic in General

## *(Prelude)*

Why this programm which looks obsolete by all these big developement environments you can download or buy...

Well this can becomes in mind but if you like old computers or if the code you has was known as very well tested and bugfree but runs oly on old computers with their build in interpreter..., or if the basic software they have used doenst run on your favorite operating system. THEN...

Another reason might be that you are frustrated by all the hussle around "modern" programming languages. Ive started programming over 20 years ago, i was late at that time - i bought my first computer as i was 19 ...

After 20 years i was forced to do this thing again in Basic, cause it was written in Basic, it runs fine, was well tested and very complicate. It would be take much more time to port it to another language and much more time to check all things out then to write bastool.

So i am back and was wondering again how simple things can be done and i remember the time as i was a young boy with a sharp pocket computer in his hands (later with the C128 and the Amiga ...) All this computers where shipped with a book titled: "BASIC REFERENCE MANUAL"

BASIC... Beginers All Symbolic Instruction Code

Programming isnt how to do it in this or that language its a way of thinking - independing of the language. (its more a bit like zen - the japanese meditation way for my opinion)

If you know one language you can learn another within a short time. And if you see that a old basic interpreter only uses a handful of instructions and does things like a editor in 200 lines... (from scratch)

Today you got a 500 MB monster of a Basic from a big Company for free. It takes longer to read the documentation and to instal all things then to write something useful by hand.
The boss of the same company has started with a basic interpreterfor the altair later for Aplle... (done in some k of assembly) *smile*
I wrote this cause ive seen things which grew to 3 megabytes and the same thing was done with 4k 20 years ago. A good example was PONG you know the simple game with the two paddles. Done in 4 k in a lot of languages. It wasSOLD as a PLAYSTATION GAME for 100 bucks on CD last year *rofl*

## *Other Basic*

You can get Basic interpreter and Compiler for free. Most of the Veterans from dos are now abandonware and can be downloaded without charge. They run fine in a dos emulation, i use dosemu for example (of course with the dos memory limitations)

You will also get modern interpreter or compilers for free.

Some runs under different OS, especially Small Basic does it.
Even if its a bit hard to install Small Basic under Linux - it runs same code also on a Palm or your Cellphone - no joke !
The IDE is simple and it knows a lot of modern construts too. So its a good start point for beginners.

Blassic and YABASIC are fine interpreters which are worth a look Freebasic has the advantage to be nearly a QBASIC GWBASIC clone. (Thats why i use it at the moment and while it has a nice documentation)

Gambas will be nice if you would like to deal with things like you would do in Visual Basic. Mono will do .NET things And of course there are also Basic tools you may buy „Pure Basic“, „K Basic“ and others.

Just a look over the table: IBM has put their great REXX Environment to the open Source community. Its really worth a try if you wont care anymore about variables and the syntax is very near to basic.
(its the most simple powerful well designed ROBUST language i know)

The pearl php python ruby xyz guys will say same about their languages
(of course *laugh*)

So aside all us serious work :
Lets blast some alien slime, hunt the wumpus, use the force
and make a nice klingon barbecue !

Have fun Thomas

# APPENDIX

```
USAGE

bastool SWITCHES <file.bas>
             (last commandline word is the always the source file)
EXAMPLE
bastool wumpus.bas
bastool -color 1 startrek.bas
bastool -split 1 -crossref 1 eliza.bas > elizanew.bas
            (last example shows how to write the output to a file)



COMMANDLINE SWITCHES:

-bft          <bool>   turn on brute force tokenizee
-renum        <bool>   turn on renumbering
-delta       <value>   renumbering steps
-split        <bool>   split lines by :
-indent      <value>   spaces used for ident
-varren       <bool>   rename variables
-vardump      <bool>   dump vars with redim statements
-labren       <bool>   rename labels to nonnumeric if numeric
-labdrop      <bool>   remove unused labels
-color        <bool>   syntax highlighting
-repprint     <bool>   repair print statements
-insendif     <bool>   make endif statements
-colortable   <bool>   print a small color table
-dbgindent    <bool>   print ident level in front of code
-strarray     <bool>   put used strings in a array
-crossref     <bool>   do cross referencing
-debug        <bool>   debug output
-listfrom    <value>   list code beginning from line
-listto      <value>   sop codelisting by line

<bool> means boolean 0=(off/false) or 1=(on/true)
<value> means a integer number 0 or bigger
```