MMBasic Source Compression Utility

crunch.zip

This module is part of the original MMBasic library. It is reproduced here with kind permission of Hugh Buckle and Geoff Graham. Be aware it may reference functionality which has changed or is deprecated in the latest versions of MMBasic.

MMBasic Source Compression Utility CRUNCH.bas v2.4 Author: Hugh Buckle June 2013 Updated: Dec 2014 to v2.4

This suite of programs takes an MMBasic program, reduces its size and increases its execution speed by: - Eliminating all comments and blank lines - Replacing all variables with 1 or 2 character names - Removing indentation - Optionally replacing label names with 1 or 2 characters - Optionally replacing Subroutine and Function names with 1 or 2 characters - Remove unnecessary # symbols and blanks - Concatenating lines of code where possible. The need for it was suggested by TZAdvantage on The Back Shed forum as he currently does this very tedious job manually. Why would you want to Crunch a program? Well there could be several reasons: - To speed up execution. - To reduce the source size for an imbedded product. - To disguise the techniques used. This is never going to be entirely successful as the persistent user can reverse engineer the code, given enough time and inclination. But it won't be easy because the variable and label names will have been replaced with one or two character, unrelated names. Unsupported functions: - CRUNCH does not support programs with line numbers. Use MMEdit or NONUMBER.BAS in MMLib to remove line numbers before running CRUNCH. - CRUNCH does not check the code for validity. Make sure your code runs before Crunching it and test it well after wards as there is no guarantee that the Crunch process is 100% accurate. - CRUNCH v2.4 does not support MicroMite MMBasic 4.5 as it will create A and A$ as separate variables and it does not recognise the "%" and "!" variable suffixes. There are three programs in the suite: KEYWORDS.BAS which creates a couple of files of keywords which Crunch must not change: - RWord.CSV, a CSV file containing a sorted list of all of the MMBasic keywords and - RWords.txt, a list of short (1 and 2 character) alpha keywords which must not be used as substitute variable names. Input is one or more files containing the names of all the MMBasic commands, functions, obsolete commands, operators and system variables and their respective parameters. KeyWords only needs to be run when MMBasic is updated with additional Commands or Functions. CRUNCH.BAS which performs the first few stages of the crunch process. It deletes all comments, blank lines and unnecessary blanks and #, if required. It identifies and marks every variable, Subroutine name, Function name and Label in the source for future substitution with a short name, writing the modified source to a temporary file. It then passes control to

---

CRUNCHa.BAS Reads the marked up source, replaces the marked items with one- or two-character names, combines lines into multi-statement lines where possible and writes the resultant code to the output file. The short names in RWords.txt are not used as substitute names as they are MMBasic keywords (e.g. IF, TO, AS, ON, OR, PI, F1, F2, …). Changes in v2.4 of the Crunch suite of programs.CRUNCH.BAS Fixed a problem whereby some keywords were not recognised in the source. Added a Testing switch /T to set the variable Testing to True (Default: False). This causes some additional messages to be printed and suppresses deletion of temporary files. Include the /T switch in Crunch.dat if required or on the command line. Changed the variable marker from "!" to "~" as it is less likely to be used anywhere in the source.

KEYWORDS.BAS has been changed (it was called RWSRTMRG.BAS – a name which reflected how it worked, not what its purpose is) to make it easier to reliably update the list of keywords (reserved words) when MMBasic is enhanced. Up to v2.3, the command and function reserved word files contained only one reserved word per line. KeyWords v2.4 allows more than one word per line so that each line can contain the command or function and all of its parameters. So now Commands and Functions can appear in the same order as they do in the MMBasic Language Manual, one line per Command or Function making it easy to check that all changes have been incorporated. The files in this distribution are up to date with MMBasic v4.5. They are contained in: Cmd.txt (Commands), Funct.txt (Functions), ObsCmd.txt (Obsolete commands), Oper.txt (operators) and SysVar.txt (System variables). I have removed output file names and the NumberOfInputFlies parameter from the Implied Run Command for KEYWORDS.BAS and from the parameter data file (KEYWORDS.DAT). The program now sets the output file names (which are used by CRUNCH.BAS) and assumes that all file names in KEYWORDS.DAT are input files. It processes them one after the other. I have reduced the number of dots printed on the screen when KEYWORDS.BAS is sorting data and have removed them altogether when running in a DOS environment, because it all happens very quickly. Running the programs

---

KEYWORDS.BAS 2 methods: i. Using the Implied Run Command Type KEYWORDS InputFilename_1[, InputFileName_2…] e.g. KEYWORDS Cmd.txt Funct.txt Oper.txt SysVar.txt ObsCmd.txt ii. Create KEYWORDS.DAT containing one record with format: InputFilename1[,InputFileName2][,InputFileName3]… e.g. Commands.txt, Functns.txt, Operands.txt, SysVars.txt, ObsCmnds.txt Then type KEYWORDS In both methods, file names may be separated with a comma or space. Extra commas and spaces are ignored.

CRUNCH.BAS 2 methods: i. Using the implied Run Command. Type CRUNCH InputSourceFileName<.BAS> OutputFileName<.BAS> </B> </F> </L> </P> </T> e.g. Crunch MyProg test ii. Create CRUNCH.DAT containing a single record with format: InputSourceFileName<.bas> OutputFileName<.bas> </B> </F> </L> </P> </T> e.g. MyProg.bas test /L /P /F Then type CRUNCH In both methods parameters may be separated by comma or space, but not mixed - use one or the other. Switches: /B suppresses removal of unnecessary blanks and tabs in the code. /F suppresses replacement of Function and Subroutine names. /L suppresses replacement of Label names. /M suppresses creation of multi-statement lines. Original multi-statement lines are retained. /P pauses the listing at a pageful and awaits any key press before continuing. /nnn sets the maximum output line length (max 255) (default 255). Note however that this length will be ignored for statements which are longer, e.g. Single-line IF and PRINT statements. /T sets Testing=True within both Crunch.bas and CrunchA.bas (used for debugging).

One or more switches may be included in any order after the OutputFileName. For safety, InputSourceFileName must not be the same as OutputFileName. If it is, then the user is prompted for a new output file name. If the output file name exists, the user is asked if it can be overwritten. If filenames do not have an extension, .BAS is assumed. CRUNCHa.BAS is not intended to be run separately. Control is passed to it by CRUNCH after creating a number of temporary files.

Testing Progress as of 8/12/2014

- All 3 programs in the suite have been tested on in a Mono MM under MMBasic 4.5 and DOS MMBasic v4.5 under Windows 7
- None have been tested in any other environment (as I don’t have access to them).

Implementation Notes.

The biggest problem I had with Crunch was deciding how to identify variables. Labels, user defined Function and Subroutine names are easy as they always appear at the beginning of a line and each has its own strict format. Comments and stuff inside double quotes are also easily identified and eliminated, but variables... So I abandoned the idea of trying to identify variables directly and looked instead for MMBasic built-in Commands and Functions, comments, strings contained in double quotes and user defined function and subroutine names. All these are known or are easily identified. Everything left over which conforms to the rules for variable names must therefore be a variable. So the first thing to do is to create a list of built-in Commands and Functions and their parameters (such as OUTPUT, TO, THEN); there are currently more than 340 of them1. These need to be sorted in ASCII sequence so that they can be looked up quickly and efficiently in a table (array), however they all contain text so it had to be a text array. Whoa! Stop right there! A string array with over 340 elements? No way! It will take up far too much memory and there won't be room for program statements and other variables. So what to do? Fortunately TassyJim on The Back Shed forum had discussed the use of short strings (See topic: Short Strings). His solution was to PEEK and POKE in memory to reduce text space. I didn't want to do that as I thought it might make the result MMBasic version dependent but it did give me the idea of packing as many reserved words as possible in each element of an array and this reduced the size of the array from 340+ elements to 8. Each of the 8 elements contains a portion of the list in Comma Separated Variable (CSV) format and it is relatively quick and easy to find a reserved word using a two stage search; look at the first word in an element to see if it will be in there, and if it is then search through that element. Building RWord.CSV, the sorted list of reserved words, is the job of KEYWORDS.BAS, but more about that later. It currently uses one or more input files. Just specify the names in KEYWORDS.dat or on the implied run command line.

CRUNCH

There aren't as many as 340 Command names and Function names, there are a lot less than that, but if you add in all of the parameters that these built-in Commands and Functions carry with them, then there are around 340 “reserved words”. Although these additional reserved words are only reserved in the context of the Command or Function they serve, in this version of CRUNCH, I have treated them all as reserved. For instance OUTPUT is a parameter of OPEN and it is quite valid to use it as a program variable name. OUTPUT variable name is therefore not changed in this version.

Having solved the problems of size and how to identify variables, the rest is relatively straight forward, if rather long-winded. First RWord.CSV is read into an array and input, output file names and switches found, either from CRUNCH.DAT or MM.CMDLine$.

Next, if either labels or user defined function and subroutine names are to be left unchanged, it does a quick pass though the input file to build an array of the names. These will then be treated as if they were reserved words. This array must be sorted so it can be searched using a binary search and, of course, it is not known how big to dimension the array until the pass is complete so, as each name is

found, it is counted and written to a temporary file. At the end of the pass the array is dimensioned and loaded from the temporary file, sorting and removing any duplicates (which there shouldn't be) as they are read. Sorting is done by checking against others already in the array and slotting it in in ASCII sequence.

Now the main task of finding variables can start. Each line of source code is read, comments and blank lines are discarded and variable candidates unearthed. These are checked against the list of MMBasic commands and Functions and against the Labels list, if it exists. If neither scores a hit, then they are variables. Each variable in the source code is enclosed in a VariableStartMark and VariableEndMark and the marked line of code written to a temporary file (CrunchC.tmp) to await a later stage where they will be replaced with a short name. The variables themselves are also written to a temporary file (CrunchV.tmp).

There is likely to be a lot of duplication of variable names as you go line by line through the source code so, as each is found and before it is written to the temporary file, it is looked up in a stack to see if it has been found recently. If it has, then it is promoted up the stack so that the most prolific ones are retained on the stack as long as possible. If it hasn't it is added to the top of the stack and, if the stack is full, the one at the bottom is written to the temporary variable name file. This won't eliminate all duplicates, but it gets rid of a lot as variables tend to be gregarious. e.g. a=a+b

Next each line is stripped of unnecessary characters. See the list under Unnecessary Characters towards the end of this document.

If the code is required to be compressed into multi-statement lines then, before writing the code to CrunchC.tmp, existing multi-line statements are split into single lines so that when they are combined in CRUNCHa it easier to fill a record to the maximum size of 255 characters. Labels are marked so that they will always start on a new line. Singleline IF (SLIF) statements are not broken up into multiple lines so that the logic is preserved. So now we have a file of variable names, albeit with many duplicates, and a file of the source code with all the variables, labels and SLIFs marked and empty lines, comments and unnecessary spaces removed. The next stage is to bring these two together by sorting the variable list, removing duplicates, assigning a new short name to each and then substituting them in the marked source code. This is the point when Crunch runs out of memory so CRUNCH creates a parameter file and passes control to CRUNCHa.

CRUNCHa

CRUNCHa starts by reading the parameter file which gives it file names, counts, the characters which mark a variable and any relevant switches. It then reads the list of variables and creates a sorted array, dropping duplicates as it goes, by using the same slot-in method employed in CRUNCH for sorting the list of label and user-defined function/subroutine names. This array will be bigger than necessary as it is initially dimensioned to the number of variables written to the temporary file by CRUNCH. So to ensure there is enough memory for the rest of processing, it is written to file CrunchSV and then read back into an array of the final size. An earlier version just copied the array to another, but ran out of memory dimensioning the array copy when the source program being crunched had a lot of variables scattered through it, making the unsorted variable list (including duplicates) large. As the variable array is being built from the sorted file, a replacement one or two character name is added to each element, separated from its parent by a comma. Numeric and string variables each have their own sequence of replacement short names so the first numeric variable will be assigned A and the first string variable A$. Only alphabetic characters are used. Two characters should be more

than enough as each type has a choice of 26×26=676 different short names available. There are, however certain one and two alpha “words” which are not allowed as variables (e.g. AS) which would change the action of the resultant code (e.g. Pi=10 doesn't change the value of a variable called Pi. When subsequently used it is still 3.14159) so, in this version of Crunch, they have not been used as replacement values. CrunchA gets these commands from RWords.txt which, like RWord.CSV, is created by KEYWORDS.

So finally, CRUNCHa can read the marked temporary source file and replace each variable, so kindly marked for it by CRUNCH, with the new short name and concatenate lines where possible. See Concatenating lines below. Job done apart from KILLing the temporary files. When testing, the temporary files aren't killed.

Concatenating lines

By default, Crunch concatenates lines where possible and removes unnecessary characters to save space; each character removed saves one byte; each new-line replaced with a colon saves 3 bytes. When creating multi-statement lines, the following measures are taken into consideration:

- Concatenated lines must not be greater than 255 characters long (or the line length specified in the /nnn switch). If /nnn is less than the length of a statement, then it is ignored for that statement.
- Two lines may be concatenated placing a single colon between them, provided that:
- The second line doesn't start with a label.
- The second line can be completely contained within the line
- A single line IF statement must not be concatenated with the next line as that could alter the result. However it can be added on to the preceding line (as long as the preceding line is not a single line IF statement). Consider:

```
A = 4
IF A = 0 THEN PRINT "Hello" : PRINT "World"
Will print nothing. But:
A = 0
IF A = 0 THEN PRINT "Hello" : PRINT "World"
```

Will display: Hello World

- A label must appear at the start of a new line.
- A label doesn't need a colon or space between it and the following statement on the same line. e.g. the following is valid:

```
label1:Print"Hello World"
```

- The first colon on a line is given a space before it, unless it defines a label. This ensures that the identifier

preceding the colon isn't mistakenly interpreted as a label.

- Parts of a command must not be split between lines. e.g. the following is invalid:

```
SPRITE
```

```
MOVE n,x,y
```

Unnecessary characters

The following characters are optional and are removed by Crunch, except when within a quoted string.

- Blanks and tabs appearing before and after the following characters: , ( ) ; = + - * / < > \ ^ % space tab.
- Space and tab before and after a colon, except when it is the first colon on a line. If it is the first, then a space before it is not removed as this could cause the preceding identifier to be interpreted as a label.
- Space and tab at the beginning and end of a line and before and after a quoted string.
- A # symbol is optional and is removed except in the following commands and when within a quoted string:

```
INPUT #nbr, list of variables
LINE INPUT #nbr, string-varaible$
PRINT #nbr, expression
WRITE &#0091;#nbr,&#0093; expression
```

KEYWORDS

Returning now to the program that creates the CSV of reserved words. Geoff very kindly sent me the complete list of Command names and Function names in MMBasic. To this I added all of their parameters. These are contained in the .TXT files in this distribution (Cmd.txt, Funct.txt, Oper.txt, SysVar.txt and ObsCmd.txt). There isn't enough memory to be able to sort all of these reserved words in one go, so KEYWORDS reads in 200 at a time for DOS and 230 at a time for the Mono Maximite (Yes, surprisingly, the MMBasic has more memory available than DOS MMBasic), sorts them and writes them to a temporary file. At EOF all input files, KEYWORDS merges the temporary sorted files into one CSV file, with each CSV record being filled to its maximum of 255 characters. Currently they can all be fitted into 8 records. While writing RWord.csv, all 1 and 2 character alphabetic reserved words are counted and written to a separate temporary file. This file is then copied to RWords.txt, adding the count of records at the front so that CrunchA can dimension an array without having to read the whole file first.

Possible future enhancements

TZAdvantage on The Back Shed, who suggested the need for CRUNCH, would like to see a number of added features which are not in the Nov 2014 version.

- Some variables are used more than others. To speed up the resultant code, the 26 most used variables should be assigned a single character short name.
- Local variables in defined function and subroutine should have their own list of replacement short names. They can start again from A and A$ and will almost certainly be single character names.

I hope to work on these enhancements once the initial programs settle down.

Hugh Buckle

Version History

v2.4 1/12/2014 Renamed RWSRTMRG.BAS to KEYWORDS.BAS and updated it to: - Allow multiple reserved words per line in input files, separated by comma. - Print an error message if a full stop appears in the input file except when part of a System Variable e.g. MM.VER. Output files are not written if one is found. - Remove output file names from Implied Run Command and from KEYWORDS.DAT - Reduce the number of progress dots while sorting and eliminate them under DOS - Remove any blanks from input files. v2.3 11/6/2013 Added stats and progress dots. Added option to set output record length. Added /T switch to set “Testing=True” Fixed CrunchA to kill CrunchV.tmp on completion of processing. Fixed fault in identifying end of statement (failed to ignore colon inside text string). v2.2 7/6/2013 Option to compress the file by creating multi-statement lines to max 255 characters. Remove al unnecessary # symbols. v2.0 18/5/2013 Added option to remove all unnecessary spaces and tabs from crunched source. v1.01 16/5/2013 Fix to CrunchA which ran out of memory sorting large list of variables. v1.0 12/5/2013 Initial version released to The Back Shed forum .