17th January 2020 Martin Head

De-Lib Decompiler

De-Lib is a decompiler for QLiberated programs. It is still a work in progress, and as it is still in development, anything, and everything is libel to change at any time.

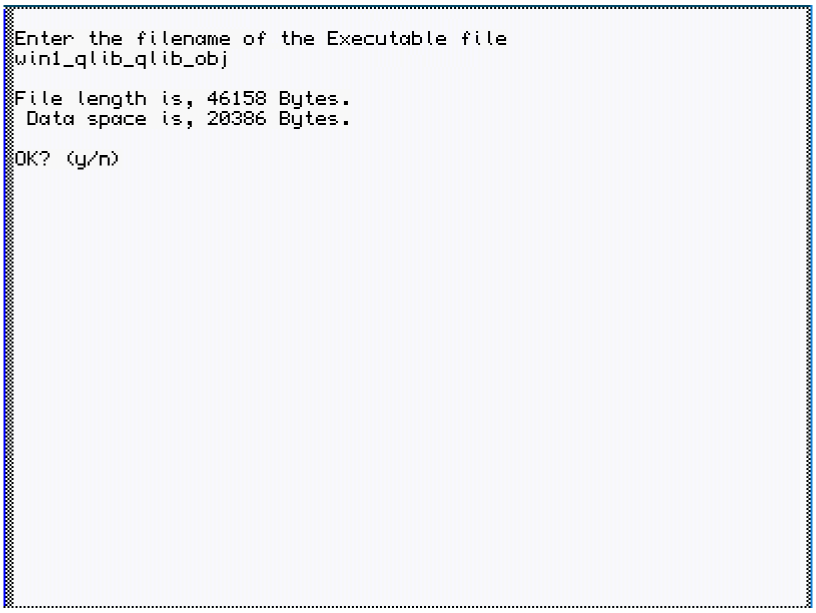
De-Lib cannot just convert an executable file back into a ready to run SuperBASIC program. It may require some manual intervention, and the resulting SuperBASIC program will need some tidying up before it will load and run correctly.

This document is an introduction to the steps required to convert a QLiberated executable back into a SuperBASIC program.

Step 1 - Do the decompilation.

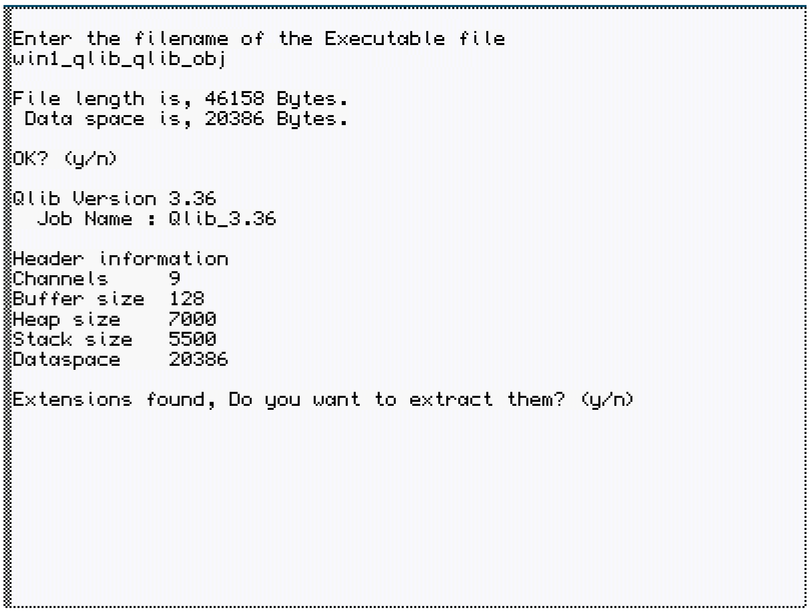
Load and **RUN** the program **DeLib_bas**.

You will be asked the name of the compiled object file to de compile. After entering the name, If the file exists, it's file size and, if the file has an intact QDOS file header, it's Dataspace size.



```
Enter the filename of the Executable file
win1_qlib_qlib_obj

File length is, 46158 Bytes.
 Data space is, 20386 Bytes.

OK? (y/n)
```

Then answer yes, or no, if it is the correct file.



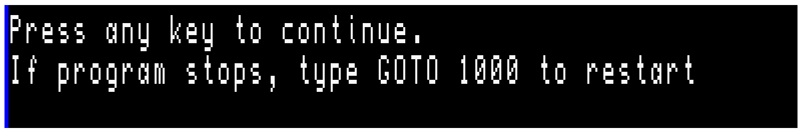
```
Enter the filename of the Executable file
win1_qlib_qlib_obj

File length is, 46158 Bytes.
 Data space is, 20386 Bytes.

OK? (y/n)

Qlib Version 3.36
  Job Name : Qlib_3.36

Header information
Channels     9
Buffer size  128
Heap size    7000
Stack size   5500
Dataspace    20386

Extensions found, Do you want to extract them? (y/n)
```

The compiled programs job name, and the version of QLiberator it was compiled on is displayed.

Information from the compiled program header area is also displayed, which may be required if you want to recompile the program at some stage, using compiler directives like **REMark $$stak=5500**

Also if DE-Lib finds any embedded SuperBASIC extensions, it will give you the option of trying to extract them from the compiled program.

In channel#0, you get the message



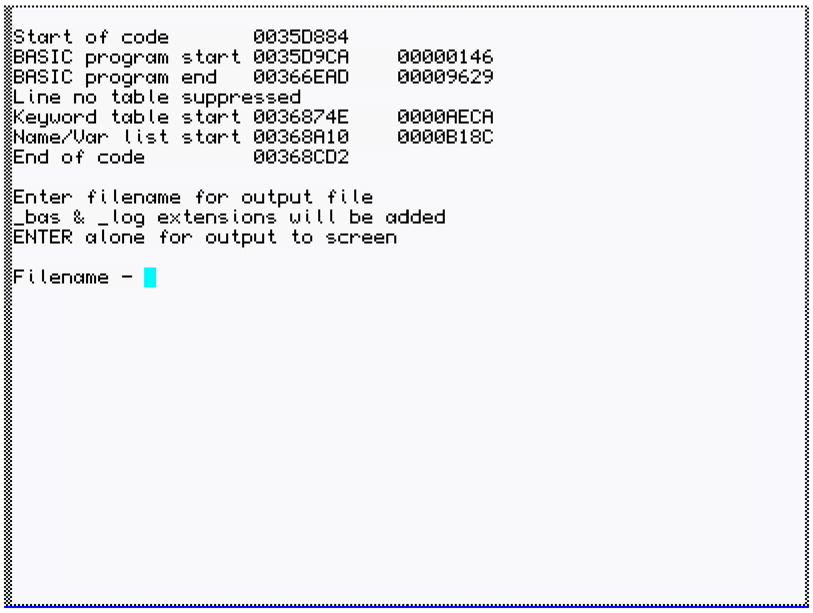
```
Press any key to continue.
If program stops, type GOTO 1000 to restart
```

You only need to **RUN** the program once, after it stops, or if you should Break into the program, use **GO TO 1000** to restart without having to go back to the beginning.

There is a program line, just past line 1000 - **pauseLine=100000**

This line allows you to pause the decompilation at a BASIC line number. And then continue one step at a time, by pressing any key. Using a number of 100000 will disable the pause, as there should not be any line numbers greater than 32767. If the compiled program has no Line number table, then line numbers greater than 32767 may be generated.

Press any key to continue

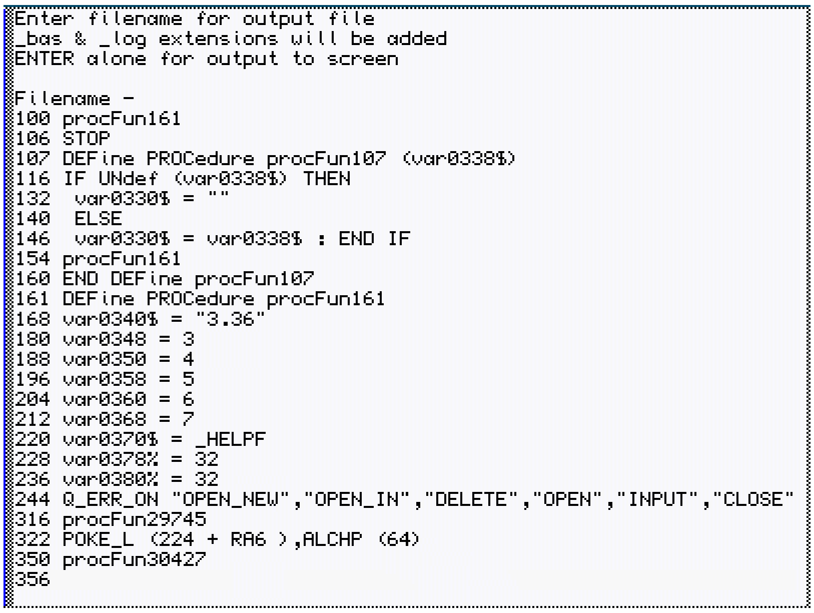


```
Start of code       0035D884
BASIC program start 0035D9CA    00000146
BASIC program end   00366EAD    00009629
Line no table suppressed
Keyword table start 0036874E    0000AECA
Name/Var list start 00368A10    0000B18C
End of code         00368CD2

Enter filename for output file
_bas & _log extensions will be added
ENTER alone for output to screen

Filename -
```

The addresses, and offsets in memory, from some of the areas of the compiled program are displayed.

When asked for a filename, just press Enter to start a decompilation to the screen. It's advisable to decompile to the screen first, just to find out if there are going to be any problems during decompiling.

The decompilation will begin up to the **pauseLine** variable, Press any key to continue one instruction at a time.



```
Enter filename for output file
_bas & _log extensions will be added
ENTER alone for output to screen

Filename -
100 procFun161
106 STOP
107 DEFine PROCedure procFun107 (var0338$)
116 IF UNdef (var0338$) THEN
132  var0330$ = ""
140  ELSE
146  var0330$ = var0338$ : END IF
154 procFun161
160 END DEFine procFun107
161 DEFine PROCedure procFun161
168 var0340$ = "3.36"
180 var0348 = 3
188 var0350 = 4
196 var0358 = 5
204 var0360 = 6
212 var0368 = 7
220 var0370$ = _HELPF
228 var0378% = 32
236 var0380% = 32
244 Q_ERR_ON "OPEN_NEW","OPEN_IN","DELETE","OPEN","INPUT","CLOSE"
316 procFun29745
322 POKE_L (224 + RA6 ),ALCHP (64)
350 procFun30427
356
```

When the decompilation stops on a **pauseLine** (in this case line 350), You can carry on, one instruction code at a time, by pressing any key. You may have to press the key several times before you see any effect on the screen.

During the decompile you may see various messages and warnings on channel#0. Most of these messages will appear in the _log file when you decompile to a file.

If DeLib encounters an instruction it does not understand. It will pause, and display the instruction code it had the problem with a yellow background.

You may be able to skip over this error and carry on by pressing any key a few times. But you may be stuck until whatever is confusing DeLib can be sorted out.

When you get a complete decompilation. Decompile again by entering **GO TO 1000.** This time entering a filename when asked. Two files will be created with your filename, and extensions of **_bas** and **_log**

The **_bas** file will be the BASIC program, and the **_log** file will contain any warnings, or errors.

If you break into the program, or it stops on an error, The variable **prog** will be near the address in the compiled program, where things went wrong, **pcount** will tell you how many items are left on the programs ‘stack’, and **PRINT getTOS$** will show the top item on the stack.

See the DeLib Technical Notes document for information on other variables and Procedures used in the program.

Step 2 - Tidy the SuperBASIC program

The SuperBASIC program produced is unlikely to Load without errors, without some tidying up. So load the produced BASIC program into a text editor. And look for some obvious problems.

Here are some of the things to look out for.

Empty program lines

If the compiled program contained a line number list, you may find program lines which are empty.

These lines may be **REMark**'s, or **SELect ON**'s, or **END IF**'s. Just add a colon (:) to these lines for now, so they will not disappear if you **LOAD** the program.

REPeat loops

QLiberator converts **REPeat** loops into **GO TO**’s.

Look out for **GO TO**’s which point back to a line right after a line containing a variable assignment of 0. Where this variable does not seem to be used anywhere.

This **GO TO** is probably the **END REPeat**.

Within this loop, If you see a **GO TO** back to the line right after the line containing the variable assignment, It is probably a **NEXT** loop. And a **GO TO** to just past the **END REPeat**, is probably a **EXIT** loop.

Here's an example

```
3100 var0698 = 0
3110 event = MCALL (#ad6)
3120
3130  [SELect] ON event = -1 : MCLEAR #ad6 : CLOSE #6 : GO TO 3240
3140  [SELect] ON event = -2 : MCLEAR #ad6 : CLOSE #6 : procFun9240  : GO TO 3240
3150  [SELect] ON event = -3 : MCLEAR #ad6 : CLOSE #6 : procFun12750  : GO TO 3240
3160  [SELect] ON event = -4 : MCLEAR #ad6 : CLOSE #6 : procFun22310  : GO TO 3240
3170  [SELect] ON event = -5 : MCLEAR #ad6 : CLOSE #6 : procFun22650  : GO TO 3240
3180  [SELect] ON event = -6 : MCLEAR #ad6 : CLOSE #6 : procFun17110  : GO TO 3240
3190  [SELect] ON event = -7 : MCLEAR #ad6 : CLOSE #6 : procFun17970  : GO TO 3240
3200  [SELect] ON event = -8 : MCLEAR #ad6 : CLOSE #6 : procFun25570  : GO TO 3240
3210  [SELect] ON event = -9 : MCLEAR #ad6 : CLOSE #6 : procFun16000  : GO TO 3240 : END SELect :
3220
3230 GO TO 3110
3240 procFun4580
```

And with the REPeat loop sorted out

```
3100 REPeat var0698
3110  event = MCALL (#ad6)
3120
3130   [SELect] ON event = -1 : MCLEAR #ad6 : CLOSE #6 : EXIT var0698
3140   [SELect] ON event = -2 : MCLEAR #ad6 : CLOSE #6 : procFun9240  : EXIT var0698
3150   [SELect] ON event = -3 : MCLEAR #ad6 : CLOSE #6 : procFun12750  : EXIT var0698
3160   [SELect] ON event = -4 : MCLEAR #ad6 : CLOSE #6 : procFun22310  : EXIT var0698
3170   [SELect] ON event = -5 : MCLEAR #ad6 : CLOSE #6 : procFun22650  : EXIT var0698
3180   [SELect] ON event = -6 : MCLEAR #ad6 : CLOSE #6 : procFun17110  : EXIT var0698
3190   [SELect] ON event = -7 : MCLEAR #ad6 : CLOSE #6 : procFun17970  : EXIT var0698
3200   [SELect] ON event = -8 : MCLEAR #ad6 : CLOSE #6 : procFun25570  : EXIT var0698
3210   [SELect] ON event = -9 : MCLEAR #ad6 : CLOSE #6 : procFun16000  : EXIT var0698 : END SELect :
3220
3230 END REPeat var0698
3240 procFun4580
```

Next we will tidy up the SELect

SELect ON

QLiberator converts **SELect**’s into **GO TO**’s, and sometimes **IT..THEN**'s internally. DeLib will try to convert them for you.

Tidying up the above example

```
3100 REPeat var0698
3110  event = MCALL (#ad6)
3120  SELect ON event
3130   ON event = -1 : MCLEAR #ad6 : CLOSE #6 : EXIT var0698
3140   ON event = -2 : MCLEAR #ad6 : CLOSE #6 : procFun9240  : EXIT var0698
3150   ON event = -3 : MCLEAR #ad6 : CLOSE #6 : procFun12750  : EXIT var0698
3160   ON event = -4 : MCLEAR #ad6 : CLOSE #6 : procFun22310  : EXIT var0698
3170   ON event = -5 : MCLEAR #ad6 : CLOSE #6 : procFun22650  : EXIT var0698
3180   ON event = -6 : MCLEAR #ad6 : CLOSE #6 : procFun17110  : EXIT var0698
3190   ON event = -7 : MCLEAR #ad6 : CLOSE #6 : procFun17970  : EXIT var0698
3200   ON event = -8 : MCLEAR #ad6 : CLOSE #6 : procFun25570  : EXIT var0698
3210   ON event = -9 : MCLEAR #ad6 : CLOSE #6 : procFun16000  : EXIT var0698
3220  END SELect
3230 END REPeat var0698
3240 procFun4580
```

Notice how the two empty lines 3120, and 3220 have now been used. Compiled programs with saved line numbers makes things easier. If the compiled program did not have line numbers saved, then you would have to add them where you think they are needed.

FOR..NEXT..END FOR statements

QLiberator uses the same code for a **NEXT** and a **END FOR** in **FOR** loops.

So if you see two **END FOR**'s in the same loop. The first one is probably a **NEXT**.

IF..THEN..ELSE
SMSQ/E can be more fussy about **IF..THEN**'s than QDOS. On **LOAD**ing the following into SMSQ/E, you may get syntax errors like. **At line 277:2 incomplete SELect clause**

```
243   ON var8978 = 99 :
247    IF ((var8938 = 0) OR var88F8%) THEN procFun1953 : ELSE
249     ....
275     ...
277     procFun1727 : END IF
279    procFun1259
```

Although SMSQ/E complains about a **SELect** clause, In this case it’s the **IF..THEN..ELSE** it’s upset about.

If you change the code to something like this -

```
243   ON var8978 = 99 :
247    IF ((var8938 = 0) OR var88F8%) THEN
248     procFun1953
249    ELSE
250     ...
275     ...
277     procFun1727
278    END IF
279    procFun1259
```

SMSQ/E will then stop complaining about it.

Another common problem with **IF..THEN** is in **REPeat** loops, to exit the loop. DeLib can have problems with an **IF THEN..EXIT** loop, As it looks exactly like a **IF..THEN..ELSE**

```
9940 var0880 = 0
9950 IF ((mark%(var0858) <> 129) OR (fil$(var0858,1) = ">")) THEN var0858 = (var0858 + 1) : GO TO
          9950 : END IF
9960 IF ((var0878 + fileng(var0858)) < (FREE_MEM  - (1024 * 10))) THEN
9970  var0878 = (var0878 + fileng(var0858))
9980  var0868 = (var0868 + fileng(var0858))
9990  ELSE
10000  var0888 = (var0858 - 1) : GO TO 10050 : END IF
10010
10020 var0858 = (var0858 + 1)
10030 IF (var0868 = total) THEN var0888 = (var0858 - 1) : ELSE
10040  GO TO 9950 : END IF
10050 FOR var03A0 = var0860 TO var0888
```

In the above example, it looks like we have a **REPeat** loop starting at line 9940, and ending on line 10040. But the **END REPeat** is in the **ELSE** part of an **IF..THEN**

When tidied up it looks like this.

```
9940  REPeat var0880
9950   IF ((mark%(var0858) <> 129) OR (fil$(var0858,1) = ">")) THEN var0858 = (var0858 + 1) : NEXT
          var0880 : END IF
9960   IF ((var0878 + fileng(var0858)) < (FREE_MEM  - (1024 * 10))) THEN
9970    var0878 = (var0878 + fileng(var0858))
9980    var0868 = (var0868 + fileng(var0858))
9990   ELSE
10000    var0888 = (var0858 - 1) : EXIT var0880
10010   END IF
10020   var0858 = (var0858 + 1)
10030   IF (var0868 = total) THEN var0888 = (var0858 - 1) : EXIT var0880 : END IF
10040  END REPeat var0880
10050  FOR var03A0 = var0860 TO var0888 STEP 1
```

Notice how the **ELSE** in line 10030 has become an **EXIT**, and the **END IF** on the end of 10040 has gone to line 10030.

What has happened is that DeLib saw a **GO TO** after the **var0888 = (var0858 - 1)**. Which looks exactly like an **ELSE**. DeLib tries to correctly identify any **GO TO**'s at the end of the True block of an **IF..THEN**, to see if it's actually an **IF..THEN..ELSE**. But **REPeat** loop **EXIT**'s can confuse it.

DEFine FuNctions
There is no easy way to determine if a FuNction in a Qliberated program returns a number, or a string. This is not a problem for SMSQ/E. But QDOS want's a '$' on the end of a string FuNction name.
If you may want to use the decompiled program in QDOS. Then you should examine all the **DEFine FuNction**s to see if they return a string. Then add a '$' to the definition, and all the calls.

Hand decompiling

If you are having problems decompiling, or you think that something has gone wrong in the decompile. Or maybe, your just interested in how it works. Then you want to decompile the compiled program manually.

These are a few brief notes to help you get started decompiling by hand.

You don't need to understand machine code to decompile by hand. A QLiberated program is just a sequence of instructions, optionally followed by some data. And making extensive use of a stack.

If you examine the DeLib Keys document, it will tell you what the instruction codes mean, and how many additional bytes are required by each instruction.

You will need to disassemble the original compiled object code, You could probably use a HEX dump instead. The DeLib Technical document gives information on the internal layout of compiled programs.

To make a better understanding of how the compiled program works. It's worth writing short BASIC test programs, and then compiling them. Then disassemble and study it.

Here are a few examples of compiled SuperBASIC program lines to get you started.

```
PAPER#4,0
02                                    We are about to do a command
   92 08034000   float 4              Put 4 on the stack
66 90            #..,                 Put a '#..,' around the item on the stack
00 92 00000000   float 0              Put 0 on the stack
66 00            no seperator         No adjustment to stack
00 96 0018       paper                Do PAPER, The 0018 is a reference to the
                                      name list


PRINT#5,20*5
02
   92 08035000   float 5
66 90            #..,
00 92 08055000   float 20             Put 20 on the stack
00 92 08035000   float 5              Put 5 on the stack
0E               multiply             Multiply two numbers on the stack
66 00            no seperator
96 0008          print



x=RESPR(1234)
00 92 080B4D20   1234                 Put 1234 on the stack
B6               respr()              Do RESPR
  D8 0028        assign x             Assign variable 0028 with value on stack



n(5,2)=100
00 92 08076400   100                  Put 100 on the stack
00 92 08035000   5                    Put 5 on the stack
00 92 08024000   2                    Put 2 on the stack
7C 02 0040       assign array         Assign the second index of array 0040
```

There are a couple of Procedures that are not used by DeLib itself. But you may find useful when hand decompiling, or just trying understand how the compiled program works.

Listnames
This Procedure will list all the names and variables in the currently loaded program to a file.
You must have decompiled the program, or started a decompile before using at as it requires certain variables to have been set.

It produces a list as shown below. The first column is the reference number you will see in the compiled program, The second column is it's type, and the third column is the way the name will appear in the decompiled program.

```
0380    Keyword                 MTEXT$
0388    Keyword                 MWDEF
0390    Keyword                 CF_STRG$
0398    Keyword                 CF_CODE
03A0    Undefined               var03A0
03A8    Integer variable        var03A8%
03B0    String variable         var03B0$
03B8    Float variable          var03B8
03C0    Float variable          var03C0
03C8    String variable         ut$
03D0    String variable         fra$
03D8    String variable         til$
03E0    String variable         pri$
```

ListLineNo
This Procedure will list all the line numbers (if they exist) in the currently loaded program to a file.

It produces a list as shown below. The first and second columns are the SuperBASIC line numbers, in decimal and hexadecimal. The third and fourth columns are the offset in decimal and hexadecimal, from the very start of the compiled program to the correct point in the compiled SuperBASIC program.

```
100     $0064   10878   $00002A7E
110     $006E   10878   $00002A7E
120     $0078   10878   $00002A7E
130     $0082   10878   $00002A7E
140     $008C   10878   $00002A7E
150     $0096   10878   $00002A7E
160     $00A0   10878   $00002A7E
170     $00AA   10878   $00002A7E
180     $00B4   10964   $00002AD4
```