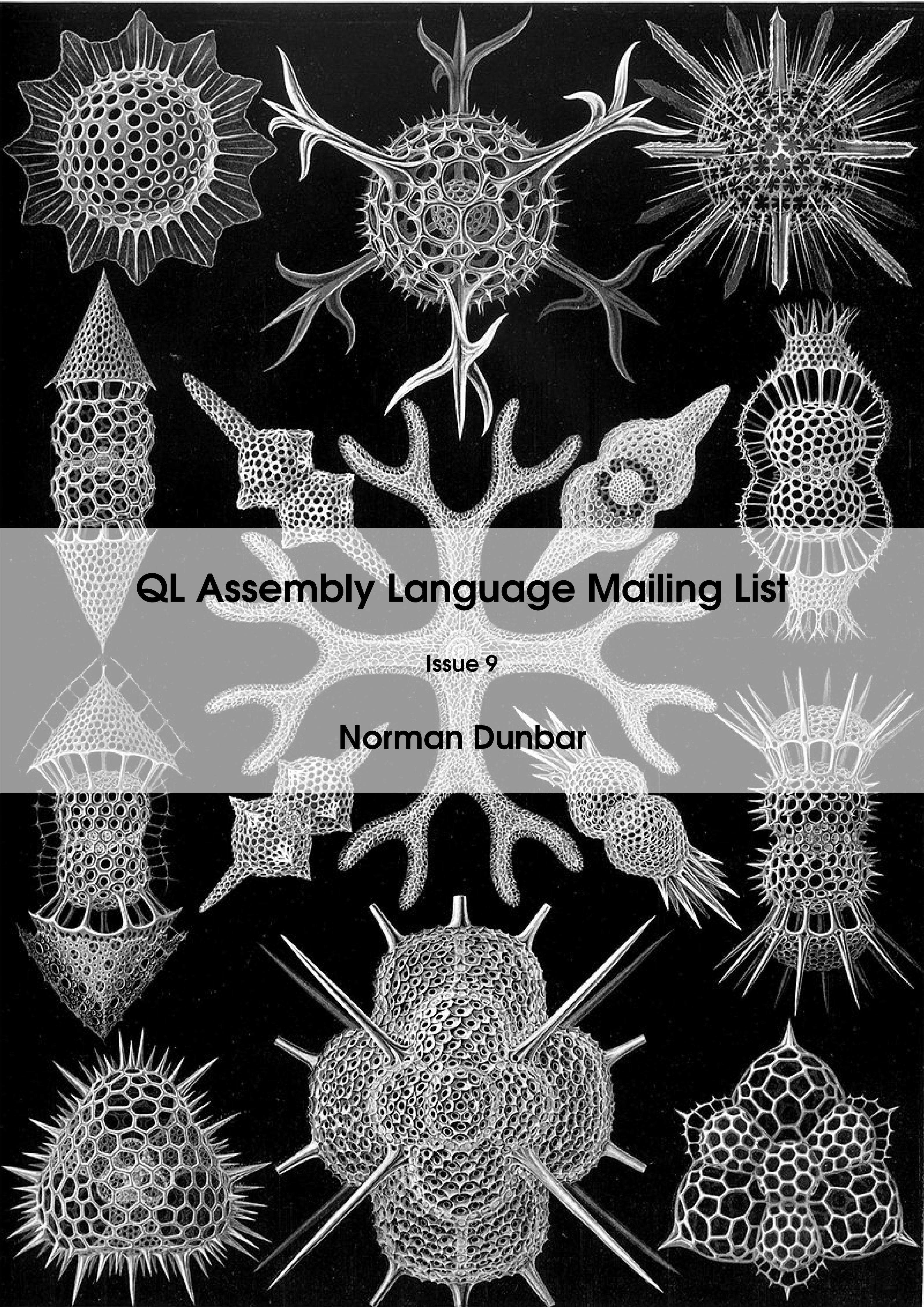


QL Assembly Language Mailing List

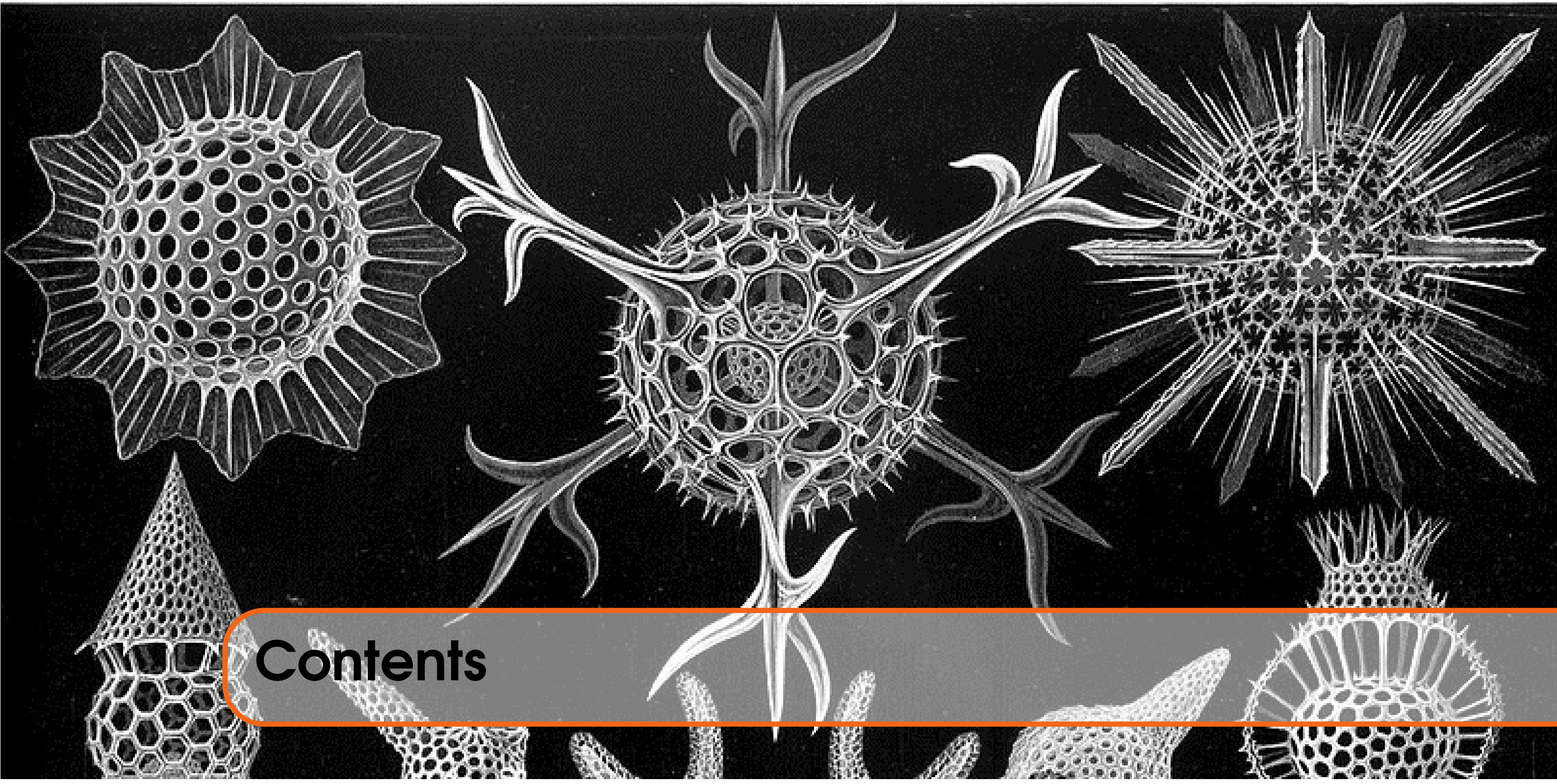
Issue 9

Norman Dunbar

PUBLISHED BY MEMYSELFEYE PUBLISHING ;-)

Download from:
https://github.com/NormanDunbar/QLAssemblyLanguageMagazine/releases/tag/Issue_9

This pdf document was created on *22/2/2021* at *13:15:58*.



Contents

Listings

1. Preface

1.1 Feedback

Please send all feedback to assembly@qdosmsq.dunbar-it.co.uk. You may also send articles to this address, however, please note that anything sent to this email address may be used in a future issue of the eMagazine. Please mark your email clearly if you do not wish this to happen.

This eMagazine is created in LaTeXsource format, aka plain text with a few formatting commands thrown in for good measure, so I can cope with almost any format you might want to send me. As long as I can get plain text out of it, I can convert it to a suitable source format with reasonable ease.

I use a Linux system to generate this eMagazine so I can read most, if not all, Word or MS Office documents, Quill, Plain text, email etc formats. Text87 might be a problem though!

1.2 Subscribing to The Mailing List

This eMagazine is available by subscribing to the mailing list. You do this by sending your favourite browser to http://qdosmsq.dunbar-it.co.uk/mailinglist and clicking on the link "Subscribe to our Newsletters".

On the next screen, you are invited to enter your email address *twice*, and your name. If you wish to receive emails from the mailing list in HTML format then tick the box that offers you that option. Click the Subscribe button.

An email will be sent to you with a link that you must click on to confirm your subscription. Once done, that is all you need to do. The rest is up to me!

1.3 Contacting The Mailing List

I'm rather hoping that this mailing list will not be a one-way affair, like QL Today appeared to be. I'm very open to suggestions, opinions, articles etc from my readers, otherwise how do I know what I'm doing is right or wrong?

I suspect George will continue to keep me correct on matters where I get stuff completely wrong, as before, and I know George did ask if the list would be contactable, so I've set up an email address for the list, so that you can make comments etc as you wish. The email address is:

`assembly@qdosmsq.dunbar-it.co.uk`

Any emails sent there will eventually find me. Please note, anything sent to that email address will be considered for publication, so I would appreciate your name at the very least if you intend to send something. If you do not wish your email to be considered for publication, please mark it clearly as such, thanks. I look forward to hearing from you all, from time to time.

If you do have an article to contribute, I'll happily accept it in almost any format - email, text, Word, Libre/Open Office odt, Quill, PC Quill, etc etc. Ideally, a LaTeXsource document is the best format, because I can simply include those directly, but I doubt I'll be getting many of those! But not to worry, if you have something, I'll hopefully manage to include it.

2. News

The news this time is that there is an actual printed copy of the *QL Assembly Language* book available. This has been created, with my blessings, by a QL Forum member named “Tinyfpga” who prefers to read paper books as opposed to PDF files. I can’t disagree!

The original, and always the latest PDF version (I keep fixing small errors, grammar, spelling etc) will always be found at:

https://github.com/NormanDunbar/QLAssemblyLanguageBook/releases/latest.

If you prefer the paper version, it’s a “print on demand” thing. “Tinyfpga” has posted these instructions on the QL Forum:

https://qlforum.co.uk/viewtopic.php?f=9&t=3294&start=40#p35207

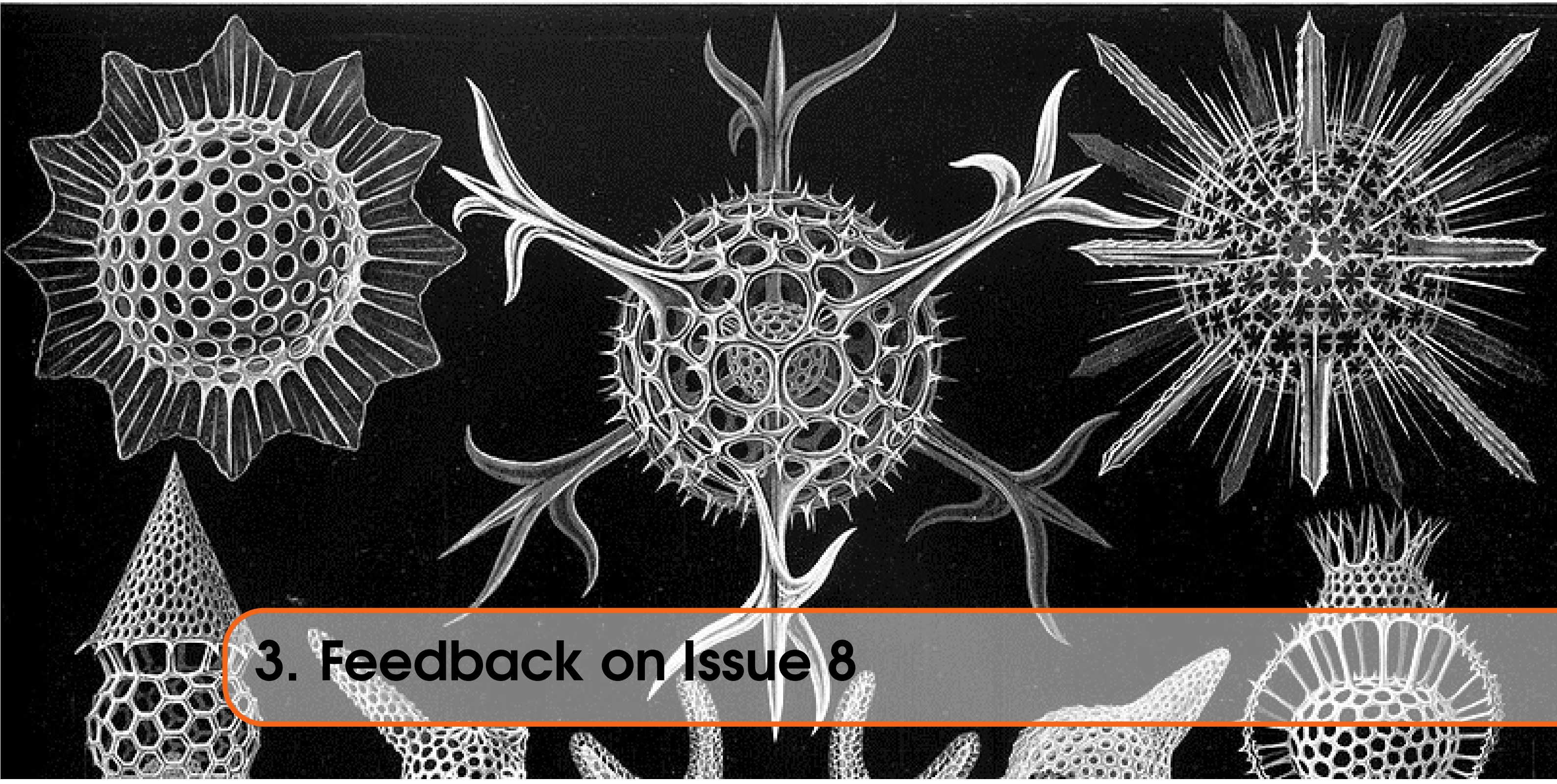
I’ve modified them slightly as it looks like the process has changed a little since first being published.

As mentioned in an earlier post I decided to "publish" (with permission) Norman Dunbar’s PDF book titled ’QL Today’s QL Assembly Language Programming Series - Book one’ as a real book.

I decided to publish it as an A4 354 page, lay-flat book for easy referencing. The book costs £7.04 plus £3.50 for postage and packaging direct from the printers. The method I have used for very low cost one-off printing to order means that anyone wanting to buy a book has to use my account to do so, as follows:

- Go to www.bookprintinguk.com
- Login as documentsforsms@gmail.com Password - forsms11 (That’s two digit ones on the end)
- Click on the Image of the book
- When the “What would you like to do” page comes up, click on the “Order More” link
- Select "order one copy" and then go next etc until you get to buy with your own name and address.

Hopefully, the process still remains the same.



3. Feedback on Issue 8

No long after release, I spotted a bug in the randomisation chapter in Issue 8. On page 49, in *Listing 7.3: Rnd 1 to 6 function - Part 1*, I had a comment on line 67 which mentioned "divide by 65536". That was complete nonsense, as that would have cleared the high word and not the low word.

Marcel spotted the error and advised me that the code was clearing overflow, not dividing. My mistake.

Marcel also spotted something else in the code I had blatantly stolen from the SMSQ sources.

It appears that the code in the rnd routine where we have this extract:

```
rnd
    ...
    mulu      #$c12d,d0            ; HHHH * 49453
    mulu      #$712d,d4            ; LLLL * 28973
  ...
    clr.w     d0                   ; HHHH 0000 (Divide by 65536)
  ...
```

Listing 3.1: Rnd 1 to 6 function

could possibly be a typo! He believes that the code should be calculating a 32 bit by 16 bit multiplication – the expression:

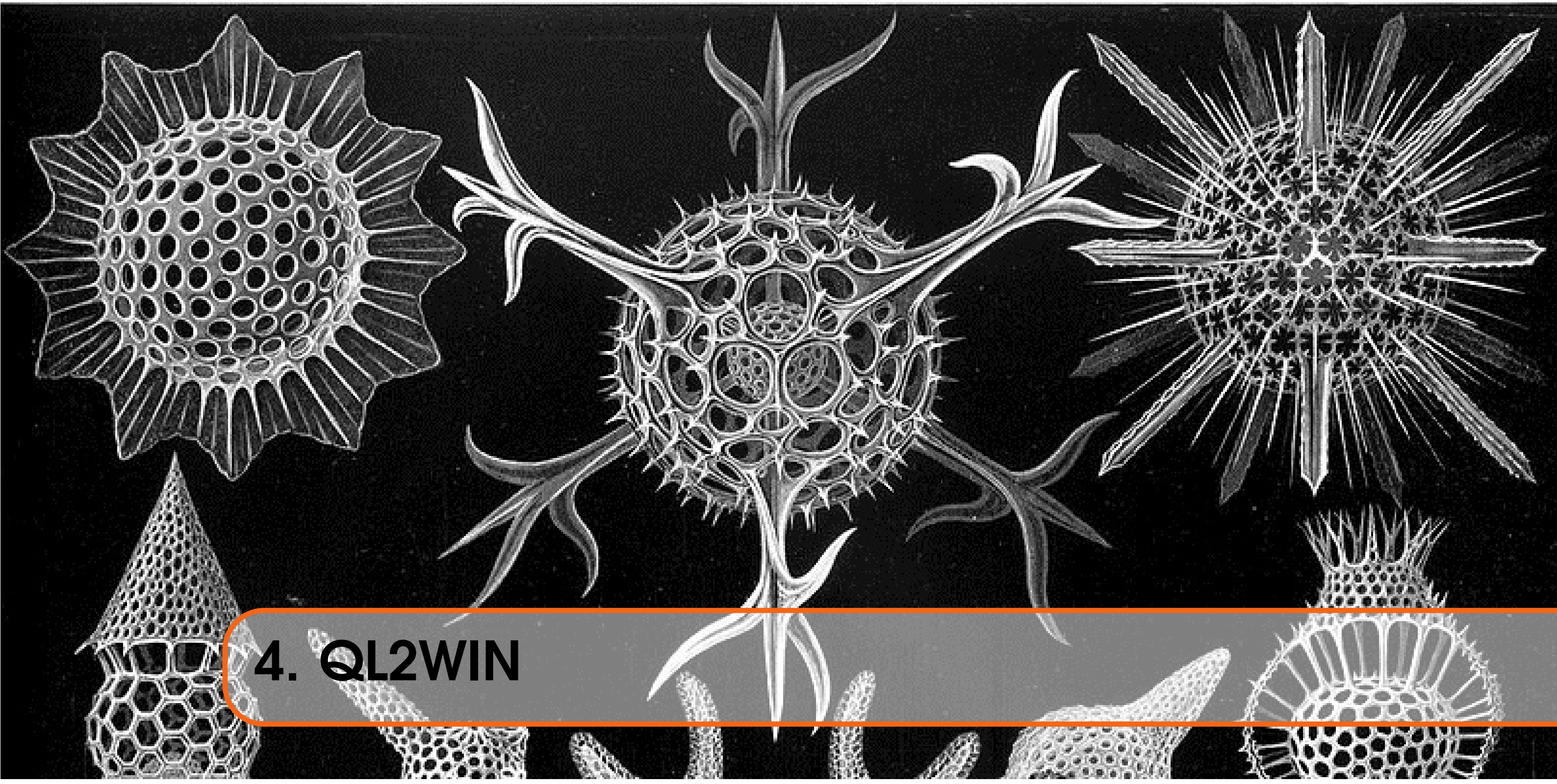
$$myRandSeed = myRandSeed * \$712D + 1$$

However, it seems to be this instead:

$$myRandSeed = myRandSeed * \$712D + ((myRandSeed \& \$F0000) * \$5000) + 1$$

Why both halves of the 32 bit random seed are not being multiplied by $712d is unknown, perhaps the used of $c12d is a typo, perhaps it's deliberate to make it more random. Nobody knows!

Thanks Marcel.



4. QL2WIN

From time to time I have to use Windows, or at least, attempt to open a file created on a Windows box while using my QL[1]. Usually, I open the file in a text editor of some kind, change the line endings setting and save the file that way, or I can use a myriad of Linux utilities to do the conversion. There are quite a few. However, this wouldn't be an ePeriodical on the use of QL Assembly Language if I didn't do it on a QL!

Given the above, I present for your wonderment and amazement, a small utility to convert a QL file to Windows format. Yes! I know! I said that I had occasion to open a Windows file on my QL, but check out the next chapter.....

4.1 The Code

It has been at least one issue, also known as "over a year", since I last wrote a YAF[2] utility. If you have missed them, then this is indeed a YAF. To convert a file from QL format with `CHR$(10)` (linefeed) line endings to Windows `CHR$(13)/CHR$(10)` (carriage return/linefeed) line endings, you simply have to:

```
EX ram1_ql2win_bin , ram1_ql_text_file , ram1_windows_txt_file
```

Listing 4.1 is the start of the code and covers a few equates and such like that I will be using through the code. As with many of my YAFs, there are only two channels required to be passed; the input QL file and the output Windows file. As I will not be faffing around in subroutines – given the extreme briefness of the code – the input channel id will be on the stack at 2(A7) while the output channel id ill be on the stack at 6(A7). The word at the top of the stack will hopefully be 2 for the number of opened channels passed.

[1] Don't ask!
[2] Yet Another Filter

```
;------------------------------------------------------------------
; QL2WIN:
;
; This filter converts QL or Linux line endings to Windows
; format.
;
; EX ql2win_bin, input_file, output_file_or_channel
;------------------------------------------------------------------
; 21/02/2021 NDunbar Created for QDOSMSQ Assembly Mailing List
;------------------------------------------------------------------
; (c) Norman Dunbar, 2021. Permission granted for unlimited use
; or abuse, without attribution being required. Just enjoy!
;------------------------------------------------------------------


;------------------------------------------------------------------
; How many channels do I want?
;------------------------------------------------------------------
numchans    equ     2           ; How many channels required?

;------------------------------------------------------------------
; Stack stuff.
;------------------------------------------------------------------
sourceId    equ     $02         ; Offset(A7) to input file id
destId      equ     $06         ; Offset(A7) to output file id

;------------------------------------------------------------------
; Other Variables
;------------------------------------------------------------------
err_bp      equ     -15
err_eof     equ     -10
me          equ     -1
timeout     equ     -1
lf          equ     $0a
cr          equ     $0d
```

Listing 4.1: Ql2win: Equates

Following on, we have Listing 4.2 which is the standard QDOSMSQ job header. There's nothing much of interest to see here, and further discussion would be fruitless. Lets move on!

```
;==================================================================
; Here begins the code.
;------------------------------------------------------------------
; Stack on entry:
;
; $06(a7) = Output file channel id.
; $02(a7) = Source file channel id.
; $00(a7) = How many channels? Should be $02.
;==================================================================
start
    bra.s   checkStack
    dc.l    $00
    dc.w    $4afb
name
    dc.w    name_end-name-2
    dc.b    'QL2WIN'
name_end    equ     *
```

```

version
    dc.w    vers_end-version-2
    dc.b    'Version 1.00'
vers_end    equ    *
```

Listing 4.2: Ql2win: Job Header

Listing 4.3 is where we check the parameters passed on the stack. We should have been passed 2 channels and a word informing us of same. The code checks that all is well, and if not, we exit with a bad parameter error.

```
;------------------------------------------------------------------
; Check the stack on entry. We only require NUMCHAN channels.
; Anything other than NUMCHANS will result in a BAD PARAMETER
; error on exit from EW (but not from EX).
;------------------------------------------------------------------
checkStack
    cmpi.w  #numchans,(a7)      ; Two channels is a must
    beq.s   ql2win              ; Ok, skip error bit

bad_parameter
    moveq   #err_bp,d0          ; Guess!
    bra     errorExit           ; Die horribly
```

Listing 4.3: Ql2win: Parameter checks

Continuing on from Listing 4.3, we have a number of constants. These are values that will be needed at various places in the code, but which are stored in spare registers to speed up the code by not having to worry about getting stuff out of buffers; or from the stack; and such like. Listing 4.4, which follows, shows that the utility is written to hold the read and write timeout in register D3, the read/write buffer size in D4, the buffer address which is used for reading and writing is held in A3 while A4 and A5 hold the channel ids for the source and destination channels respectively.

```
;------------------------------------------------------------------
; Initialise a couple of registers that will keep their values
; all through the rest of the code. These are:
;
; D3 holds the read and write timeout value, -1.
; D4 holds the buffer size for reading into, buffSize.
; A3 holds the buffer for reading and writing.
; A4 holds the source channel id.
; A5 holds the destination channel id.
;------------------------------------------------------------------
ql2win
    moveq   #timeout,d3         ; Timeout
    moveq   #buffSize,d4        ; Storage for buffer size for D2
    lea     buffer,a3           ; Start of (write) buffer
    move.l  sourceID(a7),a4     ; Source channel id
    move.l  destId(a7),a5       ; Destination channel id
```

Listing 4.4: Ql2win: Constants

These constants will be swapped into the registers that need them as the code progresses. Why bother with this? Well register to register access is much faster than memory to register access, and while it might not speed things up for the sizes of the files I use, on the odd occasions, it might be useful in bigger programs where there needs to be a lot of this sort of thing.

```
;------------------------------------------------------------------------
; The main loop starts here. Read a single byte, check for EOF.
;
; D0 = 2 (io_fline)          Error code
; D1.W                       Bytes read into buffer
; D2.W = Buffer Size         Preserved
; D3.W = timeout.            Preserved
; A0.L = Channel ID.         Preserved
; A1.L = Start of buffer.    Updated buffer (A1 + D2.W)
;------------------------------------------------------------------------
readLoop
    moveq     #io_fline,d0        ; Fetch lines ending with LF
    move.w    d4,d2               ; Buffer size
    movea.l   a4,a0               ; Channel to read
    movea.l   a3,a1               ; Read buffer start
    trap      #3                  ; Read a line from input file
    tst.l     d0                  ; OK?
    beq.s     gotLine             ; Yes
    cmpi.l    #ERR_EOF,d0         ; All done yet?
    beq       allDone             ; Yes.
    bra       errorExit           ; Oops!
```

Listing 4.5: Ql2win: readLoop

The top of the main loop for the utility is shown in Listing 4.5. Here we see the use of the `io_fline` function to read a string of bytes, from a channel, into a buffer. The string of bytes is terminated by a linefeed character, and the maximum number of bytes to be read is determined by the value in `D2.W`.

Don't do as I did, and use `io_sstrg` instead. Because that one fills the buffer regardless of where it finds a linefeed in the bytes being read. I spent ages looking for a bug in my code and had my QDOS Companion open at the `io_sstrg` page instead of `io_fline`. Sigh!

The code in Listing 4.5 sets up the registers to read from the source file and reads it. If the read was successful, we skip to the code in Listing 4.6 to process the bytes just read, otherwise we have to check for End Of File. If we find EOF, we can bale out and close the file on the way, otherwise we have an error and exit the utility via the error handling code.

```
;------------------------------------------------------------------------
; At this point, we have a string and a clean read with no
; errors. Check if we have read an entire line before we try to
; convert this to Windows format.
;
; D1.W = bytes read into buffer, inc LF.
; A1.L = one past where the LF should be.
; If -1(a1) == lf we have a whole string.
; Else write out what we have and read more of the same string.
;------------------------------------------------------------------------
gotLine
    move.w    d1,d2               ; Bytes read, required for write
    cmpi.b    #lf,-1(a1)          ; Did we read the whole line?
    bne.s     putLine             ; No, write out what we got
```

Listing 4.6: Ql2win: gotLine

The number of bytes read into the buffer is copied from `D1.W` to `D2.W` as we need `D2.W` to be correctly set for writing the bytes back out to the destination channel.

`A1.L` has been adjusted to point at the character above the trailing linefeed, if there is one, so we can check the character previous to that one. If that character is not a linefeed, then our buffer is too small to be able to read the entire line from the input channel. In this case, we simply skip to Listing 4.8 where we will write the data we have in the buffer, unchanged, to the destination channel.

If we have found a linefeed character, we drop into Listing 4.7 to process the line further, if necessary.

```
;------------------------------------------------------------
; We have read at least the end of a line and have the LF at
; the correct place in the buffer. If the character before it
; is a CR ignore it and write out, otherwise insert a CR before
; the LF and write it all out.
;------------------------------------------------------------
    cmpi.b  #cr,-2(a1)          ; Already Windows format?
    beq.s   putLine             ; Yes, ignore CR and write out
    move.b  #cr,-1(a1)          ; Insert CR
    move.b  #lf,(a1)            ; Needs LF also
    addq.w  #1,d2               ; Update count for the CR
```

Listing 4.7: Ql2win: Adding a CR

It is possible that the file we are reading is already in Windows format. Before we go ahead and write a carriage return character just before the linefeed, we better check! If the character is a carriage return, we jump off to Listing 4.8 to write the line out unchanged.

Assuming the file is not already in Windows format, we replace the linefeed at `-1(A1)` with a carriage return and then add in a new linefeed at `(A1)`. This is why we made the buffer big enough for an extra 2 characters, to give us room to add in the required carriage return.

As we have added in an additional character to the buffer, we need to update `D2.W` which currently holds the number of bytes we read in. This is used to determine how many bytes will be written to the destination file, which coincidentally enough, happens to be where we drop in next; to Listing 4.8.

```
;------------------------------------------------------------
; Write out the contents of the buffer.
;
; D0 = 7 (io_sstrg)          Error code
; D1.W                       Bytes written to channel
; D2.W = Buffer Size         Preserved
; D3.W = timeout.            Preserved
; A0.L = Channel ID.         Preserved
; A1.L = Start of buffer.    Updated buffer (A1 + D2.W)
;------------------------------------------------------------
putLine
    moveq    #io_sstrg,d0       ; Send strings
    movea.l  a5,a0              ; Dest channel id
    movea.l  a3,a1              ; Write buffer
    trap     #3                 ; Do it
    tst.l    d0                 ; OK?
    beq.s    readLoop           ; Yes, keep going
    bra.s    errorExit          ; No.
```

Listing 4.8: Ql2win: putLine

The code at `putLine` uses `io_sstrg` to write data from a buffer to a channel. The number of bytes is determined by `D2.W`. The required registers are set up by copying in those required from our constants where they have been sitting, waiting their turn of action! The remainder of the code, as seen in Listing 4.9, handles errors and exiting from the utility when all is done.

```
;------------------------------------------------------------------------
; No errors, exit quietly back to SuperBASIC.
;------------------------------------------------------------------------
allDone
    moveq    #0,d0

;------------------------------------------------------------------------
; We have hit an error so we copy the code to D3 then exit via
; a forced removal of this job. EXEC_W/EW will display the
; error in SuperBASIC, but EXEC/EX will not.
;------------------------------------------------------------------------
errorExit
    move.l   d0,d3              ; Error code we want to return

;------------------------------------------------------------------------
; Kill myself when an error was detected, or at EOF.
;------------------------------------------------------------------------
suicide
    moveq    #mt_frjob,d0       ; This job will die soon
    moveq    #me,d1
    trap     #1
```

Listing 4.9: Ql2win: Exit

There's not much to see here. We arrive at `allDone` when we hit End Of File on the input file and at `errorExit` if any errors were detected. The job then commits suicide by removing itself from the system, returning any error codes in `D3` as required. These errors will be seen only if you executed the utility with the `EXEC_W` or `EW` commands. `EXEC` and `EW` do not wait for the job to complete so cannot know in advance what, if any, errors will occur.

Finally, we have Listing 4.10, which is where we define the buffer which will be used to read data into from the source file, and write data out of to the destination file.

As previously mentioned, the buffer is two bytes larger (although it only needs one) than we tell QDOSMSQ as we need that extra one byte to insert a carriage return character, if necessary.

```
;------------------------------------------------------------------------
; Read/write buffer. The buffer is 2 bytes longer than we need
; as there needs to be room to insert the required CRLF in
; place of the LF.
;------------------------------------------------------------------------
buffSize     equ     64*2          ; Buffer Size
buffer       ds.b    buffSize+2    ; Buffer
```

Listing 4.10: Ql2win: Buffer

4.2 Filter Chains

As mentioned already, this is a YAF. It checks that you supply exactly two channels or file names on the command line and if it doesn't find exactly two, it will exit with a bad parameter error. I was thinking "what if I wanted to check my code was working and pass the output to another filter, would that work?" I just tried it out just to see.

I was working on the assumption that Tony Tebby et al, had been smart enough[3] to ensure that chains of filter programs would be set up correctly by the `EXEC_W` or `EW` commands and things would just work. My first attempt was this:

```
EX ram1_ql2win_bin, ram1_ql_text_file TO ram1_hexdump_bin, #1
```

It did indeed work as expected, the input file, `ram1_ql_text_file`, was passed into the `ql2win` filter and had carriage returns added where necessary. The output from that filter was written directly to the input channel of the `hexdump` filter, thanks to the `TO` separator, from where, the output was displayed on screen in channel 2.

This was extremely handy for testing as I could see the carriage returns added in the correct places without having to create and open additional files.

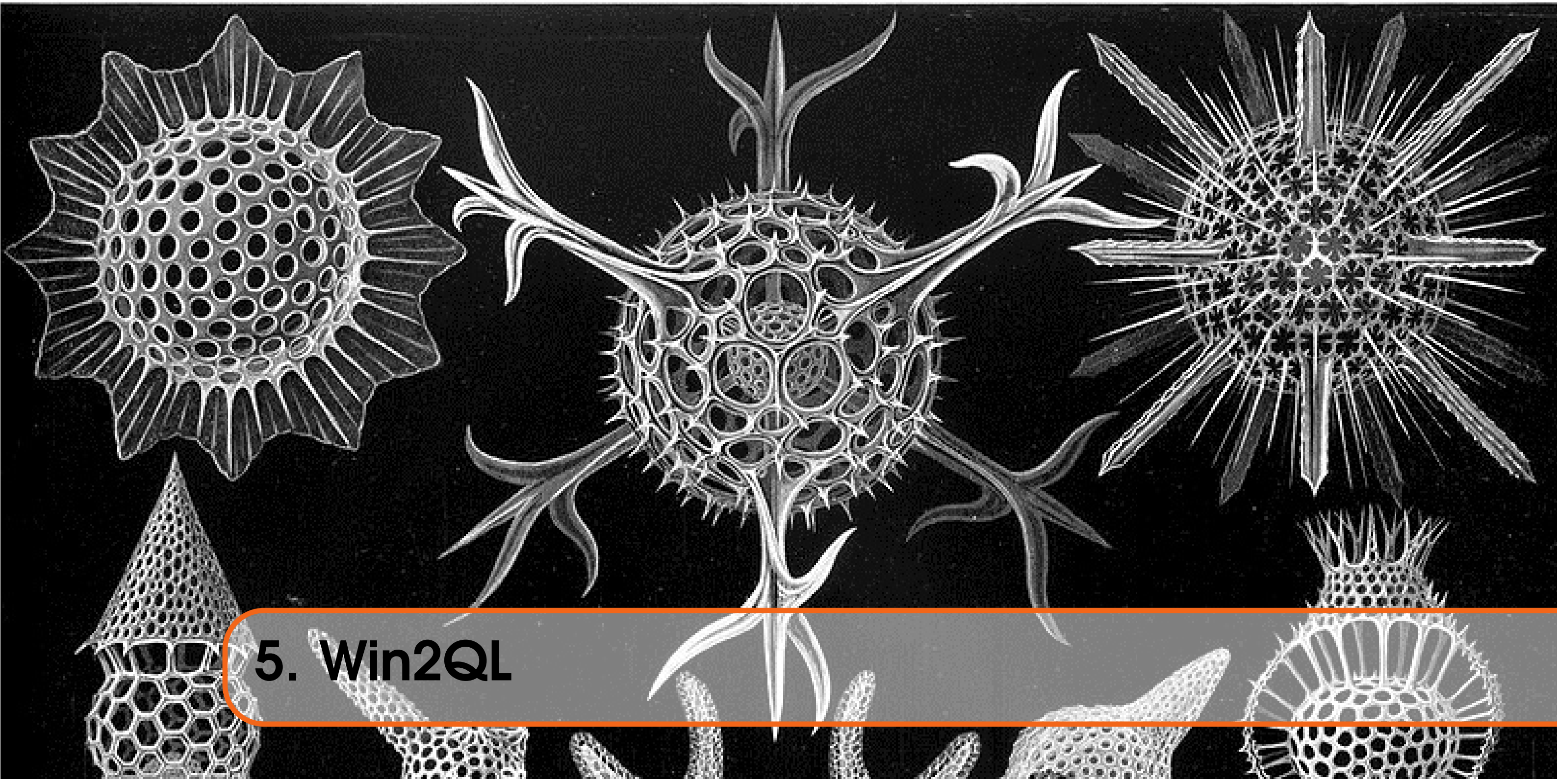
Of course, lots of YAFs can be strung together to create the final output. Here's another silly example:

```
EX ram1_ql2win_bin, ram1_ql_text_file TO ram1_win2ql_bin TO
   ⟹ ram1_hexdump_bin, #1
```

That's all one command by the way. The text file in QL format is filtered to Windows format and then passed through YAF to remove the newly added carriage returns and finally, for now, displayed on screen in hexadecimal. I used this to ensure that the output file from my two filters was identical to the text file used as the input to the test.

Once again, the `TO` separator has made sure that there are at least an input and an output channel for the filter in the chain, even though there appears to be none.

[3] And, indeed, they *were* smart enough!



5. Win2QL

So that's a utility to convert files created on the QL (or Linux!) into a format that Windows is happy with. Admittedly, even Notepad these days is able to cope with QL/Linux line endings, but it's nice to have the correct format I suppose.

Win2ql is a utility, a YAF, to convert from Windows format text files to QL format text files. It reads each line of the input file, strips off the carriage returns that it finds immediately prior to a linefeed, and writes out the adjusted buffer to the output file.

The vast majority of the code is exactly the same as discussed in the previous chapter so most of what was described there is the same and is not discussed further.

The code in the download is obviously the full utility, but for the rest of this chapter, only the changes in the file `win2ql_asm` will be discussed.

5.1 Changes From Ql2win

The first difference is in the comments at the top of the code file. Listing 4.1 in the previous chapter has been slightly amended, but only as far as the comments are concerned, none of the equates are affected. Listing 5.1 shows the new comments.

```
;--------------------------------------------------------------------
; WIN2QL:
;
; This filter converts Windows line endings to QL or Linux
; format.
;
; EX win2ql_bin, input_file, output_file_or_channel
;--------------------------------------------------------------------
; 21/02/2021 NDunbar Created for QDOSMSQ Assembly Mailing List
;--------------------------------------------------------------------
; (c) Norman Dunbar, 2021. Permission granted for unlimited use
; or abuse, without attribution being required. Just enjoy!
;------------------------------------------------------------
```

Listing 5.1: Win2ql: Comments

The next change is in the job name. Listing 5.2 shows the new job header with the amended job name. The line numbers should, *hopefully*, match those in the listings being changed, those from Ql2win.

```
;============================================================
; Here begins the code.
;------------------------------------------------------------
; Stack on entry:
;
; $06(a7) = Output file channel id.
; $02(a7) = Source file channel id.
; $00(a7) = How many channels? Should be $02.
;============================================================
start
        bra.s    checkStack
        dc.l     $00
        dc.w     $4afb
name
        dc.w     name_end-name-2
        dc.b     'WIN2QL'
name_end         equ        *

version
        dc.w     vers_end-version-2
        dc.b     'Version 1.00'
vers_end         equ        *
```

Listing 5.2: Win2ql: Job Header

There's a large gap before we hit the next change. Listing 4.7 changes to the code shown in Listing 5.3.

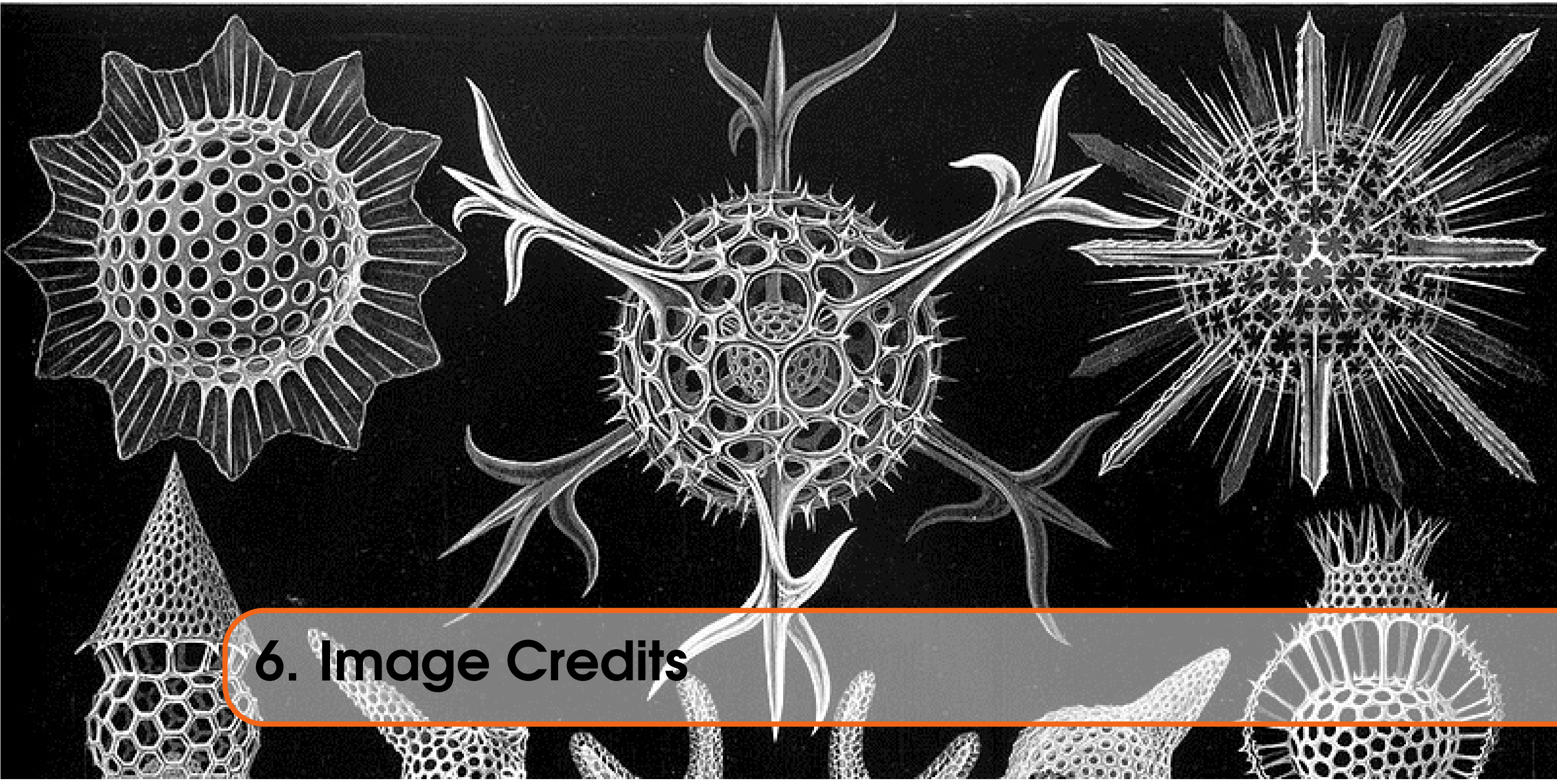
```
;------------------------------------------------------------
; We have read at least the end of a line and have the LF at
; the correct place in the buffer. If the character before the
; LF is a CR remove it and write out, otherwise just write out
; what we have, it's not in Windows format.
;------------------------------------------------------------
        cmpi.b   #cr,-2(a1)          ; Windows format?
        bne.s    putLine             ; No, write out what we have
        move.b   #lf,-2(a1)          ; Replace CR with LF
        subq.w   #1,d2               ; Update count for the missing CR
```

Listing 5.3: Win2ql: Removing a CR

As before, in Ql2win, we check if the character prior to the trailing linefeed is a carriage return. In this case, we are expecting it to be found, but if not, we can assume that this line, at least, is not in Windows format and skip off to writing the line to the output channel.

If we did find a carriage return, all we have to do is overwrite it with a linefeed and adjust the line length in `D2.W` to account for a single character less in the buffer.

The rest of the code is identical to Ql2win and has been discussed already.



6. Image Credits

The front cover image on this ePeriodical is taken from the book *Kunstformen der Natur* by German biologist Ernst Haeckel. The book was published between 1899 and 1904. The image used is of various *Polycystines* which are a specific kind of micro-fossil.

I have also cropped the image for use on each chapter heading page.

You can read about Polycystines on Wikipedia and there is a brief overview of the above book, also on Wikipedia, which shows a number of other images taken from the book. (Some of which I considered before choosing the current one!)

Polycystines have absolutely nothing to do with the QL or computing in general - in fact, I suspect they died out before electricity was invented - but I liked the image, and decided that it would make a good cover for the book and a decent enough chapter heading image too.

Not that I am suggesting, *in any way whatsoever*, that we QL fans are ancient.