TEX through `web2c` on the Mac

Timothy Murphy
(`tim@maths.tcd.ie`)

1 May 1992

Abstract

This note describes an implementation of TEX on the Macintosh, following the standard `web2c` conversion, using Symantec's THINK C compiler. It was simpler than the author (who is no guru) expected to port 'UnixTeX' to the Mac. The whole exercise is based on standard sources, the minimal changes necessary being given as `patch` files.

1 Introduction

1.1 How to get the kit

A complete 'conversion kit' is available by anonymous FTP from `ftp.maths.tcd.ie` in the directory `pub/TeX/src-5.851c/16-bit`. It is contained in the compressed TAR files `MacTeX.tar.Z` and `MacTools.tar.Z`. If you intend to do everything on the Mac you will also need `decompress.c`, `detar.c` and `detar.h`.

`MacTeX.tar` is made up of the following ASCII files (given with their approximate sizes):

`lib.patch`	37k
`web2c.patch`	14k
`web.patch`	8k
`tex.patch`	31k
`mf.patch`	38k
`mfware.patch`	5k
`MacTeX.tex`	41k

`MacTools.tar` consists of

`patch-2.0.12u6.patch`	7k
`bison-1.18.patch`	7k
`flex-2.3.7.patch`	8k

Each of the `.patch` files is a patch to be applied to the corresponding standard distribution, eg `flex-2.3.7.patch` patches the files of `flex` version 2.3.7.

1.2 Availability

All material in this directory (`pub/TeX/src-8.581c`) is in the public domain, and may be copied and modified as desired.

However, the author would be grateful to receive any corrections or suggestions for improvement, or indeed comments of any kind.

Above all, the author would very much welcome the co-operation of anyone familiar with the THINK C Class Library in adding 'Mac-ky' features to this implementation of TEX, particularly to METAFONT.

1.3 Other 16-bit systems

All the changes made to C-code are additions protected by `#ifdef THINK_C` ... `#endif`. In other words the modified files should compile on a UNIX system exactly like those in the standard distribution.

Virtually all the changes to the TEX sources would apply equally to any 16-bit system. The whole system is intended to compile as given under TURBO–C, but this pious hope has not been put to the test. (In particular, the relevant changes are actually marked by `#if defined(THINK_C) || defined(__TURBOC__)` rather than `#ifdef THINK_C`. However, this is no guarantee that they have ever compiled under TURBO–C! The author's PC is presently monopolised by 386BSD UNIX, which seemingly will not co-habit with DOS.)

1.4 The standard files

In addition to the above files, you will of course need the standard sources, to which these patches apply.

The TEX-related files (`lib`, `web2c`, `web`, `tex`, `mf` and `mfware`) are available by anonymous FTP from `ftp.cs.umb.edu` in the files `pub/tex/web.tar.Z` and `pub/tex/web2c.tar.Z`. (The `web.tar` files include TEX3.141, METAFONT2.71 and `tangle`4.3.)

The GNU software (`patch`, `flex` and `bison`) is available by anonymous FTP from `prep.ai.mit.edu` in the directory `pub/gnu`.

Please do not try to retrieve these files from `math.tcd.ie`; Ireland is linked to the rest of the world by a very thin (9600 baud) unmbilical cord.[1]

The patches will probably apply to versions near the above with few if any 'rejected hunks'. (For example, the patch for `bison-1.18` applies without change—except for the name of the folder—to the earlier version `bison-1.16`.) Normally it is easy enough to incorporate the rejected changes—which are set aside in a special file—by hand.

1.5 Mac versions

This distribution was developed and tested on a Mac Classic II with 4MB of memory, running under system 7.0.1, and using version 4.0 of the THINK C compiler.

An earlier version was tested on a MacPlus with 2MB of memory, running under system 6, and using the same compiler.

1.6 Alternative routes

For simplicity, this document describes—rather dogmatically—one way of compiling TEX and METAFONT, in which as much of the task as possible is carried on the Mac.

It will be obvious that there are many alternative scenarios. At the opposite extreme, everything could be done on a UNIX (or other) machine, up to production of the C-code for `TeXlib`, `tex`, `mf` and `mfware`. This code could then be transferred to the Mac, and compiled.

[1] My System Manager tells me that the standard GNU and TEX distributions are no longer to be found in the anonymous FTP tree!

1.7 Directory structure

The author keeps the tools in 6 top-level folders: `patch-2.0.12u6`, `bison-1.18`, `flex-2.3.7`, `decompress`, `detar` and `cdiff`. Each of these folders contains the project, and all the files, for the tool in question.

All the TEX material is in a top-level folder `TeX3.141`. The sources are in a subdirectory `src-5.851c` in this folder. (On the author's system this is the folder `Macintosh HD:TeX3.141:src-5.851c`.) We shall refer to this as the TEX source folder.

This organisation has the advantage that the standard `web2c.tar` distribution can be directly `detar`-d into the appropriate folders, using the `detar f` switch mentioned below.

Within this folder we create the following sub-folders: `web2c`, `lib`, `web`, `tex`, `mf` and `mfware`. (If you intend to `detar` into folders directly, you will also need the following folders: `bibtex`, `dviutil`, `fontutil`, `man` and `texware`, as well as the sub-folders `tex:TeXtrip`, `mf:MFwindow` and `mf:MFtrap`.)

Each THINK C project is created in the appropriate folder. Thus the 3 web2c projects—`web2c.`π, `fixwrites.`π and `splitup.`π—are all held in the `web2c` folder, the library project `lib.`π is inside the `lib` folder, etc.

The author leaves each application, when compiled, in the same folder as the project. There is in fact seldom if ever any need to drag the icon out of this folder—although of course this might sometimes be found convenient. (The author prefers to leave the applications where he knows he can find them.)

If (like `web2c`) the application requires re-direction of input, then the output will automatically be written in the same file that the input is read from. If on the other hand the program does not require re-direction (like `tangle`, say) the following trick may be found useful if you want to avoid dragging application icons around. Most of our programs do not expect input from the user after the command line has been given. Thus the standard input can be re-directed without any effect. If now it is re-directed to a file—any file—in the target folder, the effect is much the same as if the application icon had been dragged to that folder.

For example, suppose we want to apply `bison` to the file `parse.y` in the `flex` folder (as in fact we shall want to do). Then we can double click on the `bison` icon in its folder (`bison-1.18`), and now re-direct input—by pressing the `file` Radio Button under `Standard Input`—to *any* file in the

target `flex` folder. This effectively moves `bison` to this folder. We can now complete the `bison` command line

```
bison -y -d parse.h
```

and run the program.

1.8 Program size

THINK C by default allocates 375k to each application. This is insufficient for many of our programs. There are 2 ways of increasing the memory allocation.

Firstly, this can be done before compilation, by going to the `Set Project Type` item in the `Project` menu, and increasing the figure in the `Partition` box.

Alternatively, after compilation one can go to the `Get Info` item in the `File` menu, and increase the figure given in the `Memory` box in the lower right-hand corner.

For simplicity, let us agree to increase the memory to 1000k in all cases, except for TEX and METAFONT themselves, which we might increase to 1200k or more. (This is much more than needed, in most if not all cases.)

1.9 Segments

If a THINK C project contains a large number of files (or a small number of large files) it may be necessary to divide these files into 2 or more 'segments'. (A segment in THINK C can only hold about 32k of compiled code.)

This division can be done after adding the files to the project, by dragging the file-names down the project-listing. If a file-name is dragged past the end of the listing a new segment is created; if one file-name is dragged on top of another, they are placed in the same segment, as indicated by the hatching to the left.

In this note we suggest how the files might be divided into segments by using ';' as a segment-separator when listing the files. However, the actual division does not seem to matter. If there are too many files in a segment, you will get the error message '`Code overflow`'; in this case a further (or different) subdivision is required.

2 Tools

There are 6 tools in this package: `bison`, `flex`, `patch`, `decompress`, `detar` and `cdiff`.

We need the 2 GNU programs `flex` and `bison` to build TEX and METAFONT from scratch. We also need the GNU program `patch`, since we propose to define the Macintosh version of the TEX system by patches to the standard UNIX distribution.

The simplest way to transfer the standard source files to the Mac is probably as compressed TAR files, in which case `decompress` and `detar` will also be needed.

The Mac versions of `bison`, `flex` and `patch` are defined by patches to the standard GNU distributions. As indicated, the versions are `bison-1.18`, `flex-2.3.7` and `patch-2.0.12u6`. If you have versions slightly different to these, there shouldn't be too much trouble patching them. (For example, the patches given for `bison-1.18` apply without change to the earlier version `bison-1.16`.) You may get one or two 'rejected hunks', which you will have to insert by hand—but that isn't usually much problem.

The standard distributions (which you will need) can be retrieved by anonymous FTP from the GNU archive site, `prep.ai.mit.edu`, in the directory `pub/gnu`.

The patch files are all quite small, as we shall see. So there is no problem in retrieving them from us, at `ftp.math.tcd.ie`, in the directory `pub/TeX/src-5.851c/16-bit`.

The last 3 programs `decompress`, `detar` and `cdiff` are 'toys' which I got off the net, and hacked. (I gave up on GNU `tar` and `diff`.)

On the author's Mac, each of the 6 programs has its own top-level folder, with the corresponding THINK C project inside this folder.

2.1 Decompress

The simplest way to convey files to the Mac is probably as a compressed TAR file. Having brought such a file (in *binary* mode, of course) to the Mac, it must first be de-compressed.

One small problem in porting compressed text files from UNIX, and decompressing them on the Mac: UNIX does not distinguish between binary and text files, while the Mac does. The `decompress` program assumes by

default that the compressed file is binary. To de-compress a text file, the switch `-t` must be given:

```
decompress -t prog.c.Z prog.c
```

In fact, if the compressed file ends in `.Z` there is no need to name the output file:

```
decompress -t prog.c.Z
```

TAR files should be decompressed in binary mode:

```
decompress patch.tar.Z
```

2.1.1 Instructions

1. Create a `decompress` folder, and bring the file `compress.c` to it.

2. Create a THINK C project `decompress.`π in this folder. Add the file `decompress.c` to the project, and the THINK C libraries `ANSI` and `unix`. (We always add these 2 standard libraries. The former is certainly needed in most cases; but `unix` is only required for a few functions, and could probably be omitted in most cases, including this one.

3. Compile the project to create an application `decompress`.

4. Increase the memory allocated to the application, by going to the `Get Info` item in the `File` menu, and increasing the figure given in the `Memory` box in the lower right-hand corner. Let's increase the figure to 1000k, though that is certainly overkill.

2.2 Detar

The program `detar` decomposes a TAR file into its constituent files.

One advantage of porting software in TAR format is that this helps to preserve file-names against truncation imposed by some communication programs (eg `kermit`).

The version given here is minimal, with few 'bells or whistles'. There are just 2 switches: '`d`' and '`f`'. (Nb: *not* '`-d`' or '`-f`'.)

The 'd' switch gives a directory listing of a tar file:

```
tar d lib.tar
```

The extension .tar can always be omitted:

```
tar d lib
```

By default, detar will leave all the files it finds in the folder where it is run, after stripping off any directory prefix. The switch 'f' causes it to place files in the appropriate folders. However, *these folders must already exist.*

For example, the author brings Karl Berry's file web2c.tar.Z directly to the Mac, decompressing and decomposing it there, with the 'f' switch. This requires that the folders bibtex, dviutil, fontutil, man and texware already exist in the TEX source folder, as well as those listed above. In addition the sub-folders mf:MFwindow, mf:trip and tex:trap must be created before detar-ing.

2.2.1 Instructions

1. Create a detar folder, and bring the files detar.c and detar.h to it.
2. Create a THINK C project detar.π in this folder. Add the file detar.c to the project, and the THINK C libraries ANSI and unix.
3. Compile the project to create an application detar.

2.3 Patch

Suppose we have 2 versions of a file, file.old and file.new. Then the diff or cdiff program lists the differences between the 2 files:

```
cdiff file.old file.new > file.diff
```

(The program cdiff is more or less the same as diff -c on UNIX sytems.)

Now patch allows file.new to be created from file.old and file.patch:

```
patch file.old file.diff
```

Actually, the new version is called `file.old`, while the previous `file.old` is saved as something like `file.old~`. If by chance any of the changes do not appear to fit, they are saved in a file called something like `file.old#`.

There is an alternative way of running `patch`; without arguments, but with `stdin` re-directed to the patch file:

```
patch < file.diff
```

The name of the files being compared are given at the beginning of `file.diff`; and when run in this way, `patch` uses that to determine which file it is supposed to patch.

Let us be clear what we mean by this command. Having double clicked the `patch` icon, we then click the `file` Radio Button under `Standard Input`. This calls up the standard Mac 'Open File' window. Note that the file chosen may well be—usually will be—in another folder. So there is no need to drag the `patch` icon to the target folder; in effect, re-directing the standard input does this for you.

The author in fact leaves `patch` in the `patch` folder, `bison` in the `bison` folder, etc. Then he knows where to find them!

A whole collection of patch-files can be concatenated in a single file to which `patch` can be applied in the above way. For example, all the patches to the standard GNU `patch` are contained in the file `patch-2.0.12u6.patch`. So we can update all the files at once with

```
patch < patch-2.0.12u6.patch
```

(This assumes of course that the files to be patched are present in the folder.)

The port of `patch` is crude. In particular no attempt has been made to implement 'Plan A', under which the file being patched is kept in memory. (Actually, as it stands Plan A would only apply to files less than 32k in length on a 16-bit machine, so there is little loss in omitting it.)

At present `patch` simply exits (with an error message) if it cannot find the file to patch.

Also, if `patch` finds the patch must be applied 'in reverse' it does this without comment. (The UNIX version requests confirmation from the user.)

2.4 Patching patch

There is a bootstrap problem here! We need `patch` to patch `patch`. There are 3 ways out; we could do the patching on a UNIX machine, and bring

over the patched files; we can incorporate the changes by hand; or finally we can cheat and get a compiled version of `patch` to start things off—such a version can be found in the file `patch.hqx` in the same directory as the kit. We suppose in what follows that we have done one of these, and so have a working `patch` program. Perhaps we are now testing it by using it to patch `patch`.

2.4.1 Instructions

1. Create a `patch-2.0.12u6` folder, and bring the files `patch-2.0.12u6.tar.Z` and `patch-2.0.12u6.patch` to it. For simplicity let us suppose they are re-named `patch.tar.Z` and `patch.patch`.

2. Use `decompress` to decompress `patch.tar.Z`:

   ```
   decompress patch.tar.Z
   ```

   Now use `detar` to split this up

   ```
   detar patch
   ```

   (This leaves the files `patch.tar.Z` and `patch.tar`, which could be deleted now. We leave such garbage collection to the reader. It is probably advisable to leave one of these files until you are sure things have worked out properly!)

3. Delete `config.h` and copy or rename `patch-config.h` to `config.h`. (On UNIX systems, `config.h` in `patch` is obtained by running a shell script, whose output depends on the system set-up. So there is no *standard* `config.h` to base a patch on.)

4. We now suppose that you already have `patch` compiled! You could have done this by installing the patches by hand; or you may have done the patching of `patch` on another (probably UNIX) machine. In any case, now `patch patch`:

   ```
   patch < patch.patch
   ```

5. Create a THINK C project `patch.π` in this folder. Add the files `backupfile.c`, `inp.c`, `patch.c`, `pch.c`, `util.c` and `version.c` to the project, as well as the THINK C libraries `ANSI` and `unix`.

6. Compile to an application `patch`.

2.5 Bison

The GNU program `bison` is a replacement for the UNIX program `yacc`.

One curious bug (I don't think it could be called a 'feature') of THINK C—at least in version 4.0, maybe it has now been corrected—which caused the author considerable grief: `fprintf` prints the smallest integer -32768 as -1!

2.5.1 Instructions

1. Create a folder `bison-1.18`, and bring the files `bison-1.18.tar.Z` and `bison-1.18.patch` to it. For simplicity let us suppose they are re-named `bison.tar.Z` and `bison.patch`.

2. Use `decompress` to decompress `bison.tar.Z`, and `detar` to split `bison.tar` up.

3. Patch with `bison.patch`:

```
patch < bison.patch
```

4. The full path-name of the `bison` folder—or rather of the folder where the associated files `bison.simple` and `bison.hairy` are to be found—is 'hard-wired' into the file `files.c` in the `#define`s of `XPFILE` and `XPFILE1`. These 2 lines should be modified if necessary (as will probably be the case).

5. Create a THINK C project `bison.π` in this folder. Add the following C files to the project: `alloca.c`, `allocate.c`, `closure.c`, `conflicts.c`, `derives.c`, `files.c`, `getargs.c`, `getopt.c`, `getopt1.c`, `gram.c`, `lalr.c`, `lex.c`, `LR0.c`, `main.c`, `nullable.c`; `output.c`, `print.c`, `reader.c`, `reduce.c`, `symtab.c`, `version.c`, and `warshall.c`. These files should

be divided into 2 segments as indicated; those up to `nullable.c` in one segment, and those after it in the other. Finally, the THINK C libraries `ANSI` and `unix` should be added in a third segment.

6. Compile to an application `bison`, and increase the program size to (say) 1000k.

2.6 Flex

The GNU program `flex` is a replacement for the UNIX program `lex`.

We follow the recommended, but rather convoluted, way of compiling `flex`. This has 3 steps. First we use `bison` to create `parse.c` from `parse.y`. Next we copy `initscan.c` to `scan.c` and compile `flex`. Finally we use `flex` to create a new `scan.c` from `scan.l`, and re-compile `flex`.

At least this provides a stringent test of `bison` and `flex`!

2.6.1 Instructions

1. Create a folder `flex-2.3.7`, and bring the files `flex-2.3.7.tar.Z` and `flex-2.3.7.patch` to it. For simplicity let us suppose they are re-named `flex.tar.Z` and `flexn.patch`.

2. Use `decompress` to decompress `flex.tar.Z`, and `detar` to split `flex.tar` up.

3. Patch with `flex.patch`:

```
patch < flex.patch
```

4. Apply `bison` to the file `parse.y` with the switches `-y -d`:

```
bison -y -d parse.y
```

This will create 2 files, `y.tab.h` and `y.tab.c`. Rename these to `parse.h` and `parse.y`.

Note: the `bison` icon could be dragged to the `flex` folder for this. However, the author prefers the following trick—which can be played in several other places. First double-click on the `bison` icon, left at home in the `bison-1.18` folder. Now re-direct input to *any* file in the

target `flex` folder. This effectively moves `bison` to this folder. (The point is that since the application does not in fact ask for any input from the user after the command line, it does not matter where the standard input is re-directed to.) Now complete the command line as given above, and run the program.

5. Duplicate `initscan.c`, and re-name the copy `scan.c`.

6. The full path-name of the `flex` folder—or rather of the folder where the associated file `flex.skel` is to be found—is 'hard-wired' into the file `flexdef.h` in the `#define` of `DEFAULT_SKELETON_FILE`. This line should be modified if necessary (as will probably be the case).

7. Create a THINK C project `flex.`π in this folder. Add all the C files to the project: `ccl.c`, `dfa.c`, `ccs.c`, `gen.c`, `main.c`; `misc.c`, `nfa.c`, `parse.c`; `scan.c`, `sym.c`, `tblcomp.c`, and `yylex.c`. These files should be divided into 3 segments as indicated. Now add the THINK C libraries `ANSI` and `unix` as a fourth segment. Finally add to this last segment the files `alloca.c` and `xmalloc.c` from the `lib` folder in the TEX sources. (Alternatively, add `alloca.c` and `allocate.c` from the `bison` folder. Or—as a third choice—if the `TeXlib` library has been compiled, this could be added instead.)

8. Compile to an application `flex`, and increase the program size to (say) 1000k.

9. As a test of `flex`, apply it to the file `scan.l` with the switches `-ist8`, re-directing the output to `scan.c`.

10. Finally, compile this new `scan.c` and re-compile `flex`. (Note that THINK C will not realise that `scan.c` has been changed; it must be forced to re-compile.) Again increase the program-size to 1200k.

3 The TEX library

In recent versions of web2c, Karl Berry has 'hived off' the collection of supporting C-functions into a library, which we call `TeXlib`.

This is particularly convenient for THINK C, which has a facility for creating libraries. (Previously each program had its own 'extras'.)

There are a large number of files incorporated into the library; and many of these need trivial patches. Most of the 'fiddly' changes to web2c are concentrated here.

3.1 Instructions

1. We assume that you have created a folder `lib` within the TEX source folder (In my case, the folder is `Macintosh HD:TeX3.141:src-5.851c:lib`.)

   Transfer the `lib` directory of the standard distribution (`src-5.851c/lib`) to this `lib` folder.

   Transfer also the patch file `lib-5.851c.patch` to this folder as `lib.patch`, as well as the application `patch`.

2. Patch with `lib.patch`:

   ```
   patch < lib.patch
   ```

3. Delete the file `c-auto.h`, and copy or re-name `c-auto.h.in` to `c-auto.h`. (This is the same point that arose with `patch`. On UNIX systems the file `c-auto.h` is created by a shell-script, which examines the system set-up. So there is no standard `c-auto.h` to patch.)

4. Now create a project `TeXlib.`π in the `lib` folder, adding all the C-files from the `lib` directory, *except* `openinout.c` and `texmf.c`. There should be 22 files in the project: `alloca.c`, `concat.c`, `concat3.c`, `dir-p.c`, `eofeoln.c`, `file-p.c`, `fprintreal.c`, `getopt.c`, `getopt1.c`, `inputint.c`, `main.c`, `ourpaths.c`, `pathsrch.c`, `strpascal.c`, `uexit.c`. `xfopen-pas.c`, `xmalloc.c`, `xopendir.c`, `xrealloc.c`, `xstat.c`, `xstrdup.c` and `zround.c`.

5. Compile this project as a *library,* `TeXlib`.

4 The web2C suite

The program `web2c` converts the PASCAL file created by `tangle` into C. Or rather, it *almost* converts the file; the conversion of `write` statements is left to another program, `fixwrites`, which is run after `web2c`.

It may be that the resulting C program is too large for the compiler; the program `splitup` rather cleverly splits the C program into parts, with global declarations diverted to a header file which each part calls. (On the Macintosh even `tangle` requires `splitup`-ing.)

So we have to make 3 programs, `web2c`, `fixwrites` and `splitup`, for whcih we create 3 corresponding projects: `web2c.`π, `fixwrites.`π and `splitup.`π. We create these projects within the `web2c` folder; and as all take input and output by re-direction from `stdin` and `stdout`, the applications can be run from within the project; there is no need to drag them to the folder containing the files on which they are acting—as there is, for example, with `bison` and `flex`.

The 3 programs require just 6 files: `web2c.lex`, `web2c.yacc`, `web2c.c` and `web2c.h`; and `fixwrites.c` and `splitup.c`. (The file `regfix.c` in the standard distribution is not used in 16-bit TEX; it would only make sense to apply it if there were 4-byte registers available, which is not the case with THINK C or TURBO–C.)

The patch-file `web2c.patch` applies to all these files.

4.1 Web2c

Apart from the 'standard' changes expected—dynamic memory allocation with `malloc` or `calloc` for large arrays—two further modifications are worth noting.

First, before reading in a PASCAL file, `web2c` needs to be primed with a `defines` file containing 'declarations' of types and variables not declared explicitly in the program (eg functions from the standard C library). The standard `defines` file invoked by all the TEX and METAFONT prgrams is `common.defines` in the `lib` folder.

In the UNIX version, `Makefile` calls `cat` to concatenate `common.defines` with the PASCAL file. This is impossible with THINK C (and difficult in TURBO–C) so we modify `web2c` to read in `common.defines` automatically, at the start of the program.

(The TEX and METAFONT programs require a further `defines` file, `texmf.defines`, to be read in. We implement this with an additional switch, `-i`.)

The second point worth noting, perhaps, is that we have restored the ANSI option to `web2c`, causing ANSI-style function prototyping. For some reason Karl Berry dropped this from recent versions of `web2c`, relying instead

on the widespread use of casting in the so-called `coerce.h` file. While this has the advantage of working equally well with non-ANSI C compilers, it has one fairly serious drawback when working with THINK C. This Mac compiler does not allow a variable to be case as a structure. While one could easily enough re-write the half-dozen or so cases where this arises, it seemed simpler to revert to ANSI-C. (THINK C does allow structures as function arguments and return values.)

The author would argue for the return of the ANSI option even if it were not for this; `web2c` produces quite acceptable C-code, and it seems a pity to mar it by a plethora of ugly and unnecessary casting.

4.1.1 Instructions

1. We assume that you have created a folder `web2c` within the TEX source folder (In my case, the folder is `Macintosh HD:TeX3.141:src-5.851c:web2c`.)

   Transfer the `web2c` directory of the standard distribution (`src-5.851c/web2c`) to this `web2c` folder.

   The files we need to create `web2c` are: `web2c.lex`, `web2c.yacc`, `web2c.c` and `web2c.h`; and we shall also need `fixwrites.c` and `splitup.c`.

   Transfer also the patch file `web2c-5.851c.patch` to this folder as `web2c.patch`, as well as the applications `patch`, `bison` and `flex`.

2. Patch with `web2c.patch`:

   ```
   patch < web2c.patch
   ```

   (This patches all the above files, except for `web2c.lex`.)

3. Run bison on the file `web2c.yacc` with the switches `-y -d`:

   ```
   bison -y -d web2c.yacc
   ```

   This will create 2 files, `y.tab.h` and `y.tab.c`.

4. Run `lex` on `web2c.lex`:

   ```
   lex web2c.lex
   ```

   This will create a file `lex.yy.c`.

5. Create a project `web2c.π`, adding the files `web2c.c`, `y.tab.c` and `yy.lex.c` in 1 segment, and the libraries `ANSI` and `unix` in another.

6. Compile this project as an application `web2c`, increasing the program-size to $1000k$.

4.2 Fixwrites

Traditionally, `web2c` has always left the translation of PASCAL `write` and `writeln` statements to a subsequent `fixwrites` program. This is fine for those with pipes; but may a member of the pipeless minority make a plea for the marriage of `web2c` and `fixwrites` in one program—perhaps with `splitup` added as a bonus!

4.2.1 Instructions

1. Create a project `fixwrites.π`, adding the file `fixwrites.c` and the libraries `ANSI` and `unix`.

2. Compile this project as an application `fixwrites`.

4.3 Splitup

As posted, `splitup` divides the C-code into 2000-line hunks. This is a little too much for THINK C. (Each segment in THINK C has to be $<$ 32k.) For all the programs *except* METAFONT, 1900 lines is small enough. But METAFONT requires this to be cut down to 1600 lines.

It's not clear to the author if excessive sub-division slows applications down. Perhaps the size of hunks should be controlled by a switch.

4.3.1 Instructions

1. Create a project `splitup.π`, adding the file `splitup.c` and the libraries `ANSI` and `unix`.

2. Compile this project as an application `splitup`.

5 Tangle

The `tangle` program reads a `.web` file and a `.ch` change file, and creates a PASCAL file from them.

We encounter another bootstrap problem in compiling `tangle`: we need `tangle` to create `tangle`. (Recall that we needed `patch` to compile `patch`.) To solve this, we start with a previously-constructed version of `tangle`, `tangleboot.c`. (In earlier versions of `web2c` this was a simplified and supposedly universal version of `tangle`. The present version is simply `tangle.c` as created on Karl Berry's system.)

If you have difficulty compiling `tangleboot`, a compiled version is available in BinHex-ed StuffIt format as `tangleboot.hqx`.

5.1 Instructions

1. We assume that you have created a folder `web` within the TEX source folder (In my case, the folder is `Macintosh HD:TeX3.141:src-5.851c:web`.)

   Transfer to this folder the following 4 files from the `web` directory of the standard distribution: `tangleboo1.c`, `tangleboot.h`, `tangle.web` and `tangle.ch`.

2. Patch with `web.patch`:

   ```
   patch < web.patch
   ```

   (Note: we never patch the sacrosanct `.web` files.)

3. Create a project `tangleboot.`π in the `web` folder.

   Add the file `tangleboot.c` to the project in 1 segment, and the libraries `ANSI`, `unix` and `TeXlib` (which we just created) in a 2nd segment.

4. Compile this project as an application `tangleboot`. Increase its program-size to 1200k.

5. Now use `tangleboot` to `tangle` `tangle`:

   ```
   tangleboot tangle.web tangle.ch
   ```

   This will create a PASCAL file `tangle.p`.

6. Now we use `web2c` to translate this into C:

   ```
   web2c < tangle.p > tangle.cc
   ```

   There is no need to drag the `web2c` application out of its folder. Simply double-click on its icon, and press the `Standard input` 'Radio Button' to read input from the file `tangle.p`, and similarly re-direct output to the file `tangle.cc`. (This will be created in the `web` folder, since you will have moved to that folder to find the input file.)

7. Apply `fixwrites` similarly to `tangle.c`, to create the file `tangle.c`:

   ```
   fixwrites < tangle.cc > tangle.c
   ```

   Again, there is no need to move `fixwrites` from the `web2c` folder.

8. Now split up `tangle.c`

   ```
   splitup tangle < tangle.c
   ```

   This splits `tangle.c` into `tangle1.c`, `tangle2.c`, and the 1-line file `itangle.c`. Note that `splitup` can again be run from the `web2c` folder. Since the input is re-directed from the `web` folder, the output will go to that folder too.

   Note too that the name `tangle` must be given as an argument to `splitup`, to tell it how to name the files it creates.

9. Create a project `tangle.`π in the same `web` folder.

   Add the following 3 C-files to the project, in 3 segments: `tangle1.c`; `tangle2.c`; `itangle3.c`. Add the 2 libraries, ANSI and Unix, in a fourth segment.

10. In `itangle.c`, change the single line

    ```
    #define EXTERN extern
    ```

    to

    ```
    #define EXTERN
    ```

    (This step is required when splitting any file except `tex.c` or `mf.c`. Under UNIX, `splitup` is only applied to these 2 files.)

11. Compile to an application `tangle`, increasing its program-size to (say) 1000k.

6 TEX

The big one! In fact, apart from the fact that everything takes an order of magnitude more time, this module is one of the simplest. For one thing, it only contains 3 files; `tex.web` and `ctex.ch`, from the original distribution, and the patch file `tex.patch`.

There is 1 slight complication. We have to produce 2 versions of `tex`, `iniTeX` and `TeX` (or `LaTeX`), from the same C-code. Under UNIX this is done by adding or removing a compile switch `-DINITEX`. Since such switches are not allowed in THINK C, we create a 3-line file `program.h` in the `tex` folder.

This typically contains the lines

```
#define _H_program
#define TeX
#define INITEX
```

We edit the last line according to the program under compilation.

6.1 Instructions

1. We assume that you have created a folder `tex` within the TEX source folder (In my case, the folder is `Macintosh HD:TeX3.141:src-5.851c:tex`.)

   Transfer to this folder the files `tex.web` and `ctex.ch` from the `tex` directory of the standard distribution, as well as the patch file `tex.patch`.

   Use this to patch `ctex.ch`:

   ```
   patch < tex.patch
   ```

   (We could equally well use the first `patch` method:

   ```
   patch ctex.ch tex.patch
   ```

   since we are only patching the single file `ctex.ch`.)

2. Now `tangle tex`:

```
tangle tex.web ctex.ch
```

   This will create a PASCAL file `tex.p`.

3. Now convert this file to `tex.cc` with `web2c`—but add the switch `-t` (for tex) in the resulting C-code,

```
web2c -t
```

   As with tangle, press the `Standard input File` radio button to take input from `tex.p`, and the `Standard output File` button to send output to `tex.cc`. As we might put it:

```
web2c -t < tex.p > tex.cc
```

4. Apply `fixwrites` to create the file `tex.c`:

```
fixwrites < tex.cc > tex.c
```

5. Split up `tex.c`

```
splitup tex < tex.c
```

   This splits `tex.c` into `tex1.c`, `tex2.c`, `tex3.c`, ..., as well as files `itex.c` and `itex1.c`. (A function goes into these last 2 files if it differs according as `virTeX` or `iniTeX` is being compiled, ie if it contains a line `#ifdef iniTeX`.)

6. Create a project `tex.`π in the `tex` folder.

   Add the C-files `tex?.c`, `itex*.c` just created to the project, each in its own segment. Add the files `openinout.c` and `texmf.c` from the `lib` folder in another segment. Finally add the 3 libraries, `ANSI`, `Unix` and `TeXlib`, in a fourth segment.

7. Make sure there is a file `program.h` in the `tex` folder containing the following 3 lines.

```
#define _H_program
#define TeX
#define INITEX
```

8. Compile to an application `iniTeX`, increasing its program-size to 1200k or more.

   Now change the last line in `program.h` to read

   ```
   #undef INITEX
   ```

   and compile again, calling the resulting application `TeX` or `LaTeX` (according to taste). Remember to increase its memory allocation to 1200k or more.

All is now ready to install the TEX input files, and the necessary `.tfm` font files, to create the required format files `TeX.fmt` and/or `LaTeX.fmt`. But that is a matter for another document, and another day.

However, if you have the `tex` input files in place, try running

```
iniTeX plain
```

or

```
iniTeX lplain
```

(But in this note we are assuming that TEX installation is left until all is compiled.)

7 METAFONT

The other big one! This is more or less identical to TEX, as far as creation and compilation are concerned.

But this time, make sure there is a file `program.h` in the `mf` folder containing the following 3 lines.

```
#define _H_program
#undef TeX
#define INIMF
```

to compile `iniMF`. Then change the last line to

```
#undef INIMF
```

to compile MF.

If the `MFinputs` files are in place—but recall that we are not really concerned with this second phase of TEX and METAFONT installation here—then we can use `iniMF` to create a base file:

```
iniMF plain
```

After reading in `plain.mf` and other files, the prompt `*` should appear. If you have the file `modes.mf` give the response

```
*input modes
*dump
```

The file `plain.base` will be created. Re-name it `MF.base` and move it to the `bases` folder (as specified in `site.h`).

Now—assuming we have the necessary `.mf` files—we can test METAFONT by double-clicking on the `MF` icon, and giving a command like

```
MF \ mode:=epsonlq;mag=1;input cmr10
```

This should create the file `cmr10.180gf`.

8 GFtoPK

The last stage!

METAFONT outputs font files in GF format, while the vast majority of TEX driver programs require PK fonts. The program `gftopk` performs the required conversion.

The patching and compilation of `gftopk` are perfectly straightforward—and not too time-consuming, after TEX and METAFONT!

8.1 Instructions

1. Bring the 3 required files—`gftopk.web`, `gftopk.ch` and `mfware.patch` to the `mfware` folder, and carry out the patch:

```
patch < mfware.patch
```

2. Now `tangle` `gftopk`:

```
tangle gftopk.web gftopk.ch
```

3. Convert `gftopk.p` in the usual way with `web2c`

```
web2c < gftopk.p > gftopk.cc
```

4. Apply `fixwrites`:

```
fixwrites < gftopk.cc > gftopk.c
```

5. Create a project `gftopk.`π, and add `gftopk.c` to it, in one segment, and the usual 3 libraries in a second segment: `ANSI`, `unix` and `TeXlib`.

6. Compile as an application `gftopk`, and increase the allotted memory to 1200k.

7. To apply, to `cmr10.gf` say, drag this file to the folder, double click on the icon and complete the dialog

```
gftopk cmr10.gf cmr10.pk
```

9 In conclusion

As it stands, this implementation of TEX and METAFONT is little more than an exercise.

It would be nice to provide a proper 'Mac-ky' interface. In particular, it goes 'against the nature of things' to implement METAFONT on the Mac without employing the graphical facilities for which the Mac is rightly renowned.

If anyone out there—hopefully, with some experience of the Think C Class Library—is interested in co-operating on such a project (read: *doing the difficult bits*), please contact me!