FunnelWeb Hacker's Manual

Ross N. Williams

V1.0 for FunnelWeb V3.0

May 1992

Contents

Preface

This manual is for hackers! Anyone who wants to bash, diddle, frob, grind, mangle, patch, poke, toggle, twiddle, zap, or generally hack FunnelWeb should at least take a look at this manual.

This document has been created to serve two purposes:

1. To act as a repository for specific design and implementation information not appropriate to be included in program comments.

2. To guide those interested in modifying the program.

As a result, this document has grown in fits and starts. This manual is not a particularly polished or well-balanced document but it should be of assistance to those (including myself) involved with modifying FunnelWeb.

Ross N. Williams
Adelaide, Australia
May 1992

Acknowledgements

Many thanks to **David Hulse** (`dave@cs.adelaide.edu.au`) for translating the original version of FunnelWeb (FunnelWeb V1) from Ada into C and getting it to work on Unix and a PC. The C code written by David (FunnelWeb V2) formed the basis of FunnelWeb V3, but was entirely rewritten during the intensive refinement and feature-injection period leading up to this release (FunnelWeb V3 is about three times the size of FunnelWeb V2). Nevertheless, without this important first translation step, I would probably not have found the motivation to develop FunnelWeb to its present state.

Thanks go to **Simon Hackett** (`simon@internode.com.au`) of Internode Systems Pty Ltd for the use of his Sun, Mac, and PC, for assistance in porting FunnelWeb to the Sun and PC, and for helpful discussions.

Thanks go to **Jeremy Begg** (`jeremy@vsm.com.au`) of VSM Software Services for the use of his VAX, and for assistance with the VMS-specific code.

Thanks to **Barry Dwyer** (`dwyer@cs.adelaide.edu.au`) and **Roger Brissenden** (`rjb@koala.harvard.edu`) for trying out FunnelWeb Version 1 in 1987 and providing valuable feedback.

Thanks to Donald Knuth for establishing the idea of literate programming in the first place.

Ross N. Williams
Adelaide, Australia
May 1992

Presentation Notes

References: All references are set in bold and are cited in square brackets in the form [<*firstauthor*><*year*>]. All references cited in the text appear in the reference list and the index.

Special terms: New or important terminology has been set in bold face and appears in the index.

Typesetting: This document was prepared by the author using Andrew Trevorrow's (`akt150@cscgpo.anu.edu.au`) implementation (OzTeX) of the TeX/LaTeX**[Knuth84]** **[Lamport86]** typesetting system running on a Macintosh-SE.

Graphics: All diagrams have been constructed out of text using the LaTeX `verbatim` environment so as to allow this document to be disseminated electronically and printed using LaTeX without access to the author's drawing tools.

Known typesetting problems: While every attempt has been made to give a good presentation within the time available, some shortcuts have had to be taken. In particular, the author has been unable to work out how to get LaTeX to suppress blank pages at the start of chapters.

Chapter 1

FunnelWeb Design

This chapter contains notes on the design of FunnelWeb. These notes were not created "from scratch", as the original version of FunnelWeb (**FunnelWeb V1**) was designed and constructed in a hurry late in 1986, and no design notes were ever recorded. These notes are in fact the result of a complete review of the FunnelWeb design that took place in late 1991 as part of the process of upgrading it for public release.

Throughout the design process I tried to stick to the principles of simplicity and clarity. As a rule, it was considered more important that a feature be simple and not allow the user to outsmart himself than it was for the feature to be particularly crisp. For example, the FunnelWeb macro calling syntax is not as crisp as the C syntax, but is more visible.

To some extent the design review was influenced by the requirements of backward compatibility. During the review it was hard not to think about all the source files in FunnelWeb V1 format that I have written over the years that would have to be converted were I to significantly deviate from the old input language. Luckily it turned out that there was little clash between these interests, and the resulting design is both clean and does not require much conversion of old source files. The main revision is in the document structuring facility. FunnelWeb V1 had just two levels of heading indicated by `@*@*` (for major headings) and `@*` (for minor headings). This scheme (which was copied from Knuth's Web) has been replaced by a hierarchical scheme of five heading levels levels (`@A`...`@E`).

This chapter is rather unstructured, acting as it has, mainly as a dumping ground for random ideas about FunnelWeb.

1.1 Motivation for FunnelWeb

During 1986, I was exposed to Donald Knuth's `WEB` literate programming system in the form of Jon Bentley's *Programming Pearls* column in *Communications of the ACM*[**Bentley86**].[1] This prompted me to obtain a copy of the report on the Web system[**Knuth83**] and to try out the program which had been installed on the local Vaxen.

Web was the best system that I had seen for producing printed and online, inline documentation. To me the most extraordinary aspect of the system was its success despite the fact that it had been built into the horribly antiquated file/batch processing systems that we all know and love (and use). I had imagined sophisticated documentation systems before this time, but had always assumed that they would be parts of complex programming environments. Knuth showed that, to some extent, it can be done using 1960s software technology (excluding the 1980s typesetting technology). This was exciting.

[1] As it happens, this was somewhat of a lucky encounter. I had only just joined the ACM and the May 1986 issue of CACM in which the column appeared was the first issue I received.

The Web system was enticing and promising but to me suffered from many drawbacks, many of which Knuth had promoted as advantages. The following highly subjective list of disadvantages formed a springboard for the construction of FunnelWeb.

- Web can only process Pascal programs.
- Web can produce only one output file. In many instances it is desirable to generate more than one output file. For example, when programming in Ada, it is desirable to write a package specification and a package body together in the same file.
- Web enforces Knuth's individual style of indentation. Web supplies commands to over-ride the automatic indentation but it is an uphill battle and the code becomes clogged up with format directives.
- Web does not cater for non-standard Pascal programs. In particular, all identifiers are truncated to about eight characters.
- Web formats the program output file into a form that is unreadable to humans.
- Web does not provide an include facility. This was considered a feature essential for supporting macro libraries.
- Web provides macros with at most one parameter. Knuth describes a hack that can extract a multiple parameter macro facility from a single parameter one, but it is hardly satisfactory.
- Web does not provide conditionals.

Most of these objections boiled down to two points: that Web is far too specialized, and that Knuth's "Occam's Razor" had cut too far. What I wanted was a documentation system that employed all the same principles as Web but was far more general. The result was FunnelWeb V1, which can process programs in any language or any combination of languages at the cost of typesetting the text in `tt font`.

Originally, it was intended that FunnelWeb would be typesetter independent as well as language independent. It was intended that a format file consisting of a set of productions describing how the document file was to be formatted would be handed to FunnelWeb along with the input file. In the end, time pressures forced me to take the back door and hack up a TeX document file generator. This compromise has found its way into FunnelWeb V3.0 which is still reliant on TeX, although V3.0 at least encourages input files to be typesetter independent. It is hoped that future versions of FunnelWeb will include more sophisticated typesetting facilities.

1.2 Indentation

A macro call that does not appear at the left margin is called an **indented macro call** and seems to lead to three different alternatives for its expansion: **no indentation**, **blank indentation**, and **text indentation**. Here are examples of each kind of indentation. First the example problem.

```
@$@<Sloth@>==@{@-
Aardvark
Walrus@}

@O@<Output@>==@{@-
Zebra@<Sloth@>
Giraffe
@}
```

There are three ways that the second line of the `Sloth` macro can be indented.

No indentation:

```
ZebraAardvark
Walrus
Giraffe
```

Blank indentation:

```
ZebraAardvark
     Walrus
Giraffe
```

Text indentation:

```
ZebraAardvark
ZebraWalrus
Giraffe
```

No indentation is useful where the user wishes to deal with the output stream as a pure output stream. Blank indentation is useful when the user wishes to generate indented computer programs. Text indentation is useful where the user wishes to prefix each line of an entire macro invocation with a string. This can be useful for commenting out code (e.g. in Ada using `--`), and for prepending things like a dollar sign at the start of each line an a VAX VMS DCL script command file.

FunnelWeb V1 provided a choice of no indentation or blank indentation. The choice was made in the command line and could not be overridden.

The design questions are as follows:

1. Which of the three kinds of indentation should FunnelWeb support?
2. What should be the granularity of swapping between indentation modes?
3. Are particular indentation modes dangerous?
4. Is the presence of particular combinations of indentation modes confusing to the user?
5. How and when should the choice of indentation be specified?

All sorts of schemes were considered, including a finely grained system in which the user could specify at the point of call which indentation mode should be used for the called macro expansion.

After a lot of thought, the dominant factor that should affect the design was decided to be the *clarity* in the user's mind of the indentation facility and the *danger* associated with misunderstanding it. Here are two examples that show how easily a confusion or misunderstanding of the indenting being used can cause danger. The first example shows how blank or no indentation might be misused.

```
--Misuse of blank (and no) indentation.
--@<Sloth@>
```

Here the user has assumed that text indentation is in action and has placed an Ada comment symbol "`--`" before the invocation of the macro `@<Sloth@>` in the hope that the entire text of the macro would be prefixed by "`--`". The result could be passed by the compiler which would activate all but the first statement in the expansion of macro `@<Sloth@>`.

The next example demonstrates how text indentation could be misused.

```
--Misuse of text indentation:
a++; @<Sloth@>
```

Here the user has placed the call to `@<Sloth@>` after the incrementing of variable `a`. The result is that there is a good chance that the "`a++;`" prepended to each line of the expansion of `@<Sloth@>` will be passed by the compiler and will cause `a` to be overincremented.

These examples are not to be laughed at. It is possible that FunnelWeb will be used widely, and the problems above may cause problems in critical systems. The examples above are particularly scary because they are reflexive. One cannot simply pin the blame on one particular indentation form. A little thought reveals that the greatest danger lies in *confusion* in the user's mind. If the user is confused between text or blank indenting, problems will arise.

There seem to be two ways to solve the problem. The first is to ban all macro calls that are preceded by non-blank text. This is not a good option because there are so many cases where it is desirable to expand more than one single line macros on the same line. A second option is to eliminate one of the two forms so as to reduce the potential for ambiguity in the user's mind. I choose the latter option. Of the two forms, the clear choice for elimination is text indenting for the following reasons:

1. It actually introduces extra text which gives it an a priori potential for problems.
2. It is harder to implement and would slow down Tangle.
3. It would not be compatible with FunnelWeb V1 which uses blank indentation.

The only other decision is the level of granularity of choice between the remaining options: no indentation and blank indentation. FunnelWeb V1 allowed this choice to be made in the command line. In retrospect, this was bad design because the user might unwittingly code certain macro calls relying on one or the other mode. A better system is to allow the user to specify which mode in the input file itself. This has been done in FunnelWeb V3.

Again, to avoid confusion, it seems sensible to allow the user only one indentation mode per FunnelWeb input file. In most cases, the user will be happy with blank indentation (the default) and there will be no need for change anyway.

Decision: Implement only "no indentation" and "blank indentation". Make the choice of indentation a static attribute of a particular FunnelWeb run that is specified in the input file. This solution is the same as FunnelWeb V1 except that the choice has been moved from the command line to the input file.

1.3 Review of FunnelWeb Syntax

One of the distressing aspects of FunnelWeb V1 was its clumsy macro definition and calling syntax. Compared to (say) the C preprocessor, FunnelWeb's macro call syntax is like a freight train in a china show. During the FunnelWeb redesign, a complete review of this syntax took place to try to neaten it up. Surprisingly, the V1 syntax survived unscathed with the exception that "`==`" in macro definitions was made optional. The survival of this clumsy syntax was a product of the design goal of making everything simple, explicit, and not subject to subtle errors.

1.3.1 Review of Macro Definition Syntax

FunnelWeb V1 used a macro definition syntax that resulted in macro definitions such as the following.

```
@$@<Put out the cat@>==@{@-
Open the door
Say out
Close the door@}
```

This is messy, but I couldn't think of anything better at the time. The `@$` is necessary to cue a definition. Without it, the definition might somehow be mistaken for an invocation. The `@<` and `@>` delimit the name. The `@{` and `@}` delimit the text. The `@-` is a product of the rule "exactly the text between the `@{` and `@}`".

The only real target is the "`+=`" and "`==`" which really break all the rules and should be changed. Unfortunately I couldn't think of anything to change them to. If there was no `+=` mechanism, we could use:

```
@$@<Put out the cat@>@{
Open the door
Say out
Close the door@}
```

In fact, eliminating `+=` is thinkable because it does not appear in many of my existing FunnelWeb source files. This indicates at least that it was not needed much by myself.

A minimalist construct could be

```
@<Put out the cat@>
Open the door
Say out
Close the door@}
```

but this is too dangerous for my tastes.

Decision: For compatibility reasons, retain the old `+=` and `==` constructs, but make them optional. The new syntax for defining macros is:

```
@$@<Put out the cat@>@{
Open the door
Say out
Close the door@}
```

Next we turn to parameterized macro definitions. A conventional FunnelWeb parameterized macro definition looks like this:

```
@$@<Put out the cat@>@(@3@)==@{@-
Open the door
Say out
Close the door@}
```

which is a bit messy. The natural alternative is even worse:

```
@$@<Put out the cat@>@(@1@,@2@,@3@)==@{@-
Open the door
Say out
Close the door@}
```

Thus, just specifying the number of parameters seems sensible. However, perhaps the syntax could be trimmed to

```
@$@<Put out the cat@>@3==@{@-
Open the door
Say out
Close the door@}
```

I decided to reject this in favour of the old syntax.

Decision: Retain the old syntax of ...`@(@3@)`...

1.3.2 Review of Macro Call Syntax

Here are some ideas for alternatives to the FunnelWeb V1 macro call syntax.

```
Open the door
@<Say Out@>         @! FunnelWeb~V1 style.
Close the door

Open the door
@<Say out>@
Close the door

Open the door
@"Say out@"
Close the door

Open the door
@(Say out@)
Close the door

Open the door
@<Say out>
Close the door
```

Decision: Continue with the old notation. It may not be neat, but at least it is clear and consistent. The main temptation is the format `@<say out>@` which looks rather good. However, it breaks the special sequence rational and hence is too confusing.

1.3.3 Review of Parameterized Macro Call Syntax

FunnelWeb V1 provided a messy parameterized macro call syntax:

```
@<Say Out@>@(@"firstparam@" @, @"Secondparam@" @, @"thirdparam@" @)
```

This syntax can be cleaned up considerably by making the `@"` symbols optional. This results in calls such as the following:

```
@<Say Out@>@(firstparam@,Secondparam@,thirdparam@)
```

As the first form allows the alignment of complicated parameters by allowing white space to be inserted outside the `@"`, and the second form is cleaner, both are retained simply by making the quotes optional.

Decision: Make the double quotes optional.

1.4 Document Structuring

Experience with FunnelWeb V1, which provided only two levels of headings (major and minor) proved that there was a strong need for fully hierarchical multiple-level headings. The only question was how it should be done.

Here are some ideas that were considered.

```
@*@<Main Program@>
@**@<Read the Message@>
@***@<Encrypt the Buffer@>

@*@<Main Program@>
@*@*@<Read the Message@>
@*@*@*@<Encrypt the Buffer@>

@s@<Main Program@>
@ss@<Read the Message@>
@sss@<Encrypt the Buffer@>

@s@<Main Program@>
@s@s@<Read the Message@>
@s@s@s@<Encrypt the Buffer@>

@S@<Main Program@>
@SS@<Read the Message@>
@SSS@<Encrypt the Buffer@>

@S@<Main Program@>
@S@S@<Read the Message@>
@S@S@S@<Encrypt the Buffer@>

@A@<Main Program@>           -- The syntax finally chosen.
@B@<Read the Message@>
@C@<Encrypt the Buffer@>

@A Main Program
@B Read the Message
@C Encrypt the Buffer

@*A Main Program
@*B Read the Message
@*C Encrypt the Buffer

@1@<Main Program@>
@2@<Read the Message@>
@3@<Encrypt the Buffer@>
(using @A..@I@ as macro parameters or overload @1..@9)

@*@1@<Main Program@>
@*@2@<Read the Message@>
@*@3@<Encrypt the Buffer@>
(using @A..@I@ as macro parameters or overload @1..@9)
```

Choosing between these alternatives was not easy. The following thoughts contributed to the decision.

- Syntaxes that require visual counting are probably not a good idea.
- Syntaxes that do not delimit the heading name somehow are likely to cause problems where heading names are omitted. Users will be tempted to start paragraphs after the start of heading symbol and the result is that the first line of the paragraph will be sucked into the heading.
- Overloading the `@1`, ..., `@9` sequences is undesirable.

Decision: Use `@A`...`@E` with optional following macro name syntax for the section name. Note: We stop at `@E` because five levels is probably sufficient, and we may wish later to use `@F` for **F**ile (to augment or replace `@O`).

1.5 Discussion of Some Miscellaneous Issues

Comment duplication: If the FunnelWeb user inserts comments into the target code (in the `.fw` file) as well as into the documentation (free text) part of the `.fw` file, then it is possible for the situation to get a bit silly. **Decision:** This is a problem for the programmer, not for FunnelWeb.

Out-of-date documentation: Sometimes it is all too easy for the programmer to modify the code without updating the surrounding documentation. **Decision:** This is a serious and major problem. In an automated environment, it may be possible to create a system of dependencies between scraps of code and scraps of documentation. However, it is hard to see how a tool such as FunnelWeb could provide support for prevention of this sort of error.

Meta-macro-level parameterization: Sometimes, when using FunnelWeb, the facility to use one macro to construct the name of another has been needed. **Decision:** Allowing macro names to be constructed would lose the simple nature of the preprocessor and so this suggestion is rejected.

Clumsy notation: The `@` notation can be clumsy. **Decision:** This is necessary to maintain the simplicity of the translation.

1.6 Automated Regression Testing

Automated regression testing is extremely important for two reasons:

1. It provides confidence that changes made to the program have not introduced bugs.

2. It allows portability problems to be pinpointed when the program is moved to a new machine.

The simplest way to set up automated regression testing is to construct a suite of test cases (and their solutions) and then write a script in the target machine's command language to run through the suite. Unfortunately, there is no command language that is shared among the machines to which FunnelWeb must be ported. These machines are at least: Macintosh, IBM-PC, Sun, VMS Vax.

One option is simply to rewrite the script in each machine's particular command language. This would be a feasible option were it not for the fact that the Macintosh (the machine upon which FunnelWeb was developed) does not have a command language!

After some thought, I decided that the best solution to the problem was to create a command language *within FunnelWeb*. FunnelWeb could then be invoked in two modes, one-shot command line or interactive/script. This approach had the benefit of providing total control over the command language and its complete portability.

The result is described in the *FunnelWeb User's Manual*.

1.7 Command Line Interface

FunnelWeb V1 was implemented in Ada and runs on a VMS VAX. As such it has a full VMS DCL command line interface. Here is the "`.CLD`" file for the DCL command line interface.

```
! FUNNELWEB Command Definition
! ============================
! Ross Williams. 28 April 1987.

module command_table
   define verb dummy_command
   parameter p1           , label=input_file   ,value(required,type=$file)
   qualifier include_files, label=include_files,value(type=$file) ,default
   qualifier output_files , label=output_files ,value(type=$file) ,default
   qualifier delete       , label=delete                          ,default
   qualifier tex_file     , label=tex_file     ,value(type=$file) ,default
   qualifier listing_file , label=listing_file ,value(type=$file) ,default
   qualifier brief        , label=brief,value(type=$number,default=5),default
   qualifier screen       , label=screen       ,value(type=$number,default=0)
   qualifier trace        , label=trace
   qualifier file_spec    , label=file_spec
   qualifier compare
```

As portability was a key goal of FunnelWeb V3, it was obvious that the command line interface would have to be redesigned. The design goals for the new command line interface were:

1. The interface should not *depend* on case. However, it must allow case-sensitive filenames to be transmitted on systems that have case-sensitive filenames.
2. Each option must have a symmetric positive and negative form. For example, it is confusing for `-X` to turn on a feature and `-Q` to turn it off.
3. The interface must be extensible to allow inclusion of more features at a later date.
4. Options should be consistent and memorable.

The result is described in the *FunnelWeb User's Manual.*

1.8 File Name Management

File names present a host of problems for a program like FunnelWeb. First, FunnelWeb can generate so many different kinds of files that conventions must be adopted to prevent them from becoming unmanageable. Second, the constraints on file names, and even the structure of file names themselves varies considerably from machine to machine. These two problems have combined to result in the sophisticated and rather complicated way in which FunnelWeb V3 handles filenames.

To summarize, the three problems are:

1. What filename extensions should be chosen for various kinds of file?
2. What filename inheritance should take place?
3. How should FunnelWeb cope with the variations in filename structure between machines?

The following three sections address these questions.

1.8.1 Filename Extensions

FunnelWeb is capable of reading and writing a variety of different kinds of files. In particular, FunnelWeb must often operates in an environment where the same information is stored in many forms (e.g. prog.fw, prog.c, proc.exe). File extensions are an essential tool in managing this situation. The filename extensions chosen for FunnelWeb are:

```
FunnelWeb     : .fw
Product       : None.
Documentation : .tex
Listing       : .lis
Journal       : .jrn
```

Lowercase will be used in systems that are case sensitive.

Readers who are wondering how FunnelWeb copes in environments such as UNIX where there are no file extensions should refer to Section 1.8.3.

1.8.2 Filename Inheritance

Inheritance in filenames refers to how input and output files inherit parts of their name from other filenames and their environment. For example if the command

```
fw sloth +J +L +T
```

was issued, you would expect to see output files `sloth.jrn`, `sloth.lis`, and `sloth.tex`. The output file names have inherited the "`sloth`". The following table gives the hierarchy devised for FunnelWeb.

	Script	Input	Include	Journal	List	Document	Product
1			`@i`				`@o`
2	`+x`	`+f`	`+i`	`+j`	`+l`	`+t`	`+o`
3	"`.fws`"	"`.fw`"	"`.fwi`"	"`.jrn`"	"`.lis`"	"`.tex`"	
4			`+f`	`+f`	`+f`	`+f`	
5	DefDir	Defdir	Defdir	Defdir	Defdir	Defdir	Defdir

The following notes explain the table.

1. This scheme is similar to that used in FunnelWeb V1.
2. The journal, list, and documentation files all fall in the same pattern and can be considered as a single case.
3. Level 1 has the highest priority because it is a direct specification by the user in the input file.
4. Level 2 comes next because this is also a direct specification from the user on the command line.
5. Level 3 provides the default file extensions. Product files do not inherit an extension as they could be of any type.
6. Level 5 is built into most operating systems' file specification systems. If I specify file "`x.y`", it is taken to mean on the default disk in the default directory.

7. Level 4 looks straightforward, but secretly conceals a difficult design decision. By the time we get down to this level of inheritance, we know for sure that the filename has already picked up a file extension. So all that is left to inherit is the path and the filename proper. Obviously we have to inherit the filename proper (e.g. `sloth` in `sloth.tex`), but should we inherit the input file path? If we do inherit the input file path, files will be placed in the same directory as the input file. If we don't inherit the input file path, files will be placed in the current directory. The choice I have made is to send all the logging type files into the same directory as the input file. This means, for example, that `sloth.lis` and `sloth.tex` will generally land in the same directory as `sloth.fw`. However, I have decided that output files should be sent to the default directory (if not earlier specified) as this is where the action is. In normal use, the main product of FunnelWeb will be product files and so the user will expect them to appear in the current directory by default.

1.8.3 Portable Structure of File Names

Another problem with file names is the variation of their structure between environments. Here are examples of some of the formats that prevail:

```
UNIX   /device/dir1/dir2/name
VMS    node::device:[dir1.dir2]name.ext;vn
MSDOS  device:\dir1\dir2\name.ext
MAC    device:dir1:dir2:name
```

Isn't it amazing that none of these popular systems use the same format?

The solution to dealing with these different formats is to classify them as non-portable and hide the functions that manipulate them in the machine-specific module of FunnelWeb. Luckily there are not many such functions.

The main problem is coping with file systems that do not explicitly support file extensions. With so many possible input and output files, FunnelWeb all but needs such extensions. Machines that do not support them pose difficult design decisions. If the user specifies "`sloth`" as an input file on such a non-extension-supporting system, should FunnelWeb look for `sloth` or `sloth.fw`? If the user specifies `walrus` as a listing file, should it generate `walrus` or `walrus.lis`?

Some possible solutions are:

1. Regard the filename `sloth` as having an empty extension. It will then default to `sloth.fw`.

2. Regard the filename `sloth` as having a blank but full extension. That is, it cannot be overwritten by inheritance, but it remains blank.

3. Provide an extra syntactic mechanism to allow the user to specify one or other of the two options above.

My solution was to choose the first option. Use of FunnelWeb results in lots of files lying around (e.g. `sloth.lis`) and it is hard to see how the user will cope with them all without some kind of naming discipline. If a naming discipline has to be used, it might as well be the FunnelWeb one.

Thus the names of all files read and written by FunnelWeb will have a file extension of from zero to three letters separated from the rest of the filename by a "`.`".

The only exception is product files whose extension is left undefined. Product files need not contain a "`.`" and a file extension, although they can inherit one if the user wishes.

1.9 Specifying Constraints on the Number of Instantiations

Experience with FunnelWeb V1 demonstrated the need to be able to specify in macro definitions how many times it was expected that the macro would be used. FunnelWeb V1 generates an error if a macro is not used, but permits macros to be called more than once. This caused problems for macro libraries, which would be included, but whose macros were often not called.

By default, FunnelWeb V3 requires that each macro (except for the ones attached to output files) be called exactly once. However, it also provides syntax that allows the user to specify that a macro be allowed to be called zero times or many times. This allows a macro to be specified with the following permissible ranges of numbers of calls depending on the presence or absence of "`@Z`" and "`@M`":

```
0..1       @$@<Sloth@>@Z==...
   1       @$@<Sloth@>==...
   1..n    @$@<Sloth@>@M==...
0.....n    @$@<Sloth@>@Z@M==...
```

The only two problems with this scheme are:

1. It is incompatible with FunnelWeb V1 files, as the default in FunnelWeb V1 is $1 \ldots n$ whereas the default in new FunnelWeb is 1. This is not a big problem because most macros in the old files were used exactly once. Only a few macros will have to be changed.
2. What should the syntax be? (above is a sneak preview only!)

The initial proposal for syntax was to allow the user to insert zero, one, or both of `@?` and `@M` just after the `@$` of a macro definition. However, this has the following drawbacks:

1. It uses two sequences that are desirable to reserve (`@?` for conditionals and `@M` for macro).
2. It stops the user from searching for the string `@$@<name` to find the definition of a macro.

These are significant problems. Here are some alternative ideas for where to position the modifiers in the macro definition:

```
@?@M@$@<Slothy dogs@>@(@5@)==@{@-
This is a short macro.
With only a line or two@}

@$@<Slothy dogs@>@?@M@(@5@)==@{@-
This is a short macro.
With only a line or two@}

@$@<Slothy dogs@>@(@5@)@?@M==@{@-
This is a short macro.
With only a line or two@
```

The first form puts me off because I think that it is a good visual rule to start all the macros with `@$`. The second form puts me off because it detaches the macro name from the parameter list, thus making it look less like a call, which is desirable syntactic resonance. The third form is messy but probably workable. Because we are right next to the tested constant string (either `==` or `+=`) we could augment it further. For example, `01==` could allow a macro to be called from 0 to 1 times. The main problem with this is that we are trying to phase out `==` anyway!

Nevertheless, all the logic points to after the parameter list as the best place to locate this information.

After some thought, it was decided that the "`@?`" sequence be reserved for a possible future conditional facility, and so `@Z` was used instead.

Decision: The position is after the parameter list. The notation is `@Z` for zero, and `@M` for many.

Example of final syntax:

```
@$@<Slothy dogs@>@(@5@)@Z@M+=@{@-
This is a short macro.
With only a line or two@}
```

1.10 The Relationship Between Document Structure and Macro Structure

Having already decided upon a fully hierarchical document structure, I determined to refine the details. The issues to be addressed were as follows:

- How should the hierarchical structure connect to the macro structure?
- How can backwards compatibility be achieved? Should it?
- Should the macros be cross referenced by section or by definition?
- Should nameless sections inherit macro names as headings?
- Should we simply use TeX macros to structure the document?

After some thought, I arrived at the following thoughts:

Basically a TeX file: One option is simply to treat each `.fw` file as a TeX file laced with macros. That is, a `.fw` file could be structured as a real TeX file from which FunnelWeb could extract macro definitions. This approach is feasible because TeX could be programmed to respond to the `@` sequences in the same way that FunnelWeb responds to them. Thus, there would be no need for Weave.

I rejected this approach, first because it is too typesetter-dependent, and second because it complicates the inclusion of any sort of complicated post-processing in the documentation file. This would have to be implemented in TeX.

A better approach is to use an invented section notation (e.g. `@A @B @C`). This maintains FunnelWeb's typesetter independence and can easily be converted into TeX sectioning macros by Weave.

In the same spirit, it might be worth introducing a few `@` sequences for certain general typesetting operations such as italics and program code.

Confusion in FunnelWeb V1 Heading Numbering: FunnelWeb V1 uses two levels of section headings, but numbers all the headings sequentially. In this, it is a little confused. Clearly with a fully hierarchical document structure, the headings cannot be numbered sequentially — the numbering must reflect the structure (e.g. 3.2.1).

Hierarchical numbering is messy for macros: Unfortunately, hierarchical numbering is messy and confusing when applied to macro names. In FunnelWeb V1's typeset output, each macro call has appended in square brackets the number of the section in which the macro is defined. Use of hierarchical numbering would be somewhat messy. For example, a macro call might look like.

```
Write out the output[6.7.4.3]
```

Similarly, cross reference lists would be messy:

```
This macro is used in 3.4.5, 1.2, 7.8.9, 7.4, 2.2.1.1.
```

Separate numbering for macros and headings: One idea is to use hierarchical numbering for the sections, but to number the macros sequentially. This could be a little confusing in documents without much structure, but would be very much less messy than cross referencing using hierarchical numbering. Also, it will be easier to find macros indexed by a sequential number than by section, which has a less direct relationship with page bulk and number. By macro numbering is meant the sequential numbering of each macro body part through the whole document.

Input format matters more than output format: At this point we realize that a distinction should be made between the *input format* and the *typeset output*. The critical issue here is not how the program should be formatted for printing, but rather the format of its `.fw` file. The typeset output can always be changed simply by fiddling with Weave. However, as soon as the document structuring features of FunnelWeb are fixed, they will be used in dozens or hundreds of documents and it will be very difficult indeed to change them. Therefore, the important thing is to provide as sensible and expressive a `.fw` format as possible.

It is therefore a separate decision as to whether we should number macros by section or by sequence number. The important thing is to address the format and rules for the expression of structure.

Naming sections: The naming of sections requires some thought. In many cases (especially in the case of high-level sections) the writer will provide an explicit name for a section. In other cases, provision of such a name will merely duplicate the name of the macro contained within the section. It therefore makes sense to allow the user to omit the name from a section, with Weave naming the section after the first macro definition in the section. If a macro is unnamed and there is no macro in the section, an error can be generated.

All these thoughts lead to the following scheme:

- Documents will be hierarchically structured using `@A`, `@B` etc.
- Each section can be given a name delimited by `@<@>`.
- Sections that do not have names inherit the name of their first macro.
- If a section does not have a name or a macro, it is erroneous.
- Sections will be numbered hierarchically either by FunnelWeb or by TeX.
- Macro body parts will be numbered sequentially by FunnelWeb and cross referenced by these numbers.

All this results in a system which:

- Provides a hierarchical document structuring capability.
- Is typesetter independent.
- Does not require duplication between heading and macro names.
- Separates the heading and macro systems so that Weave can be configured at a later date to cross reference in different ways without requiring input files to be reworked.

1.11 Diagnostic Messages

In FunnelWeb, all error messages commence with an indicator indicating the severity of the error message. Here are some of the formats that I investigated before settling on the final format:

```
W--Error creating sloth.
E--Error opening output file.
S--I'm a teapot.
F--Can't open output file.

W-Error creating sloth.
E-Error opening output file.
S-I'm a teapot.
F-Can't open output file.

W:Error creating sloth.
E:Error opening output file.
S:I'm a teapot.
F:Can't open output file.

W: Error creating sloth.       -- Format chosen.
E: Error opening output file.
S: I'm a teapot.
F: Can't open output file.

War-Error creating sloth.
Err-Error opening output file.
Sev-I'm a teapot.
Fat-Can't open output file.

W-Old fashioned feature.

W-Old fashioned feature.

W--Old fashioned feature.

W: Old fashioned feature.

W:Old fashioned feature.
```

1.12 Summary

This rather unstructured chapter has addressed some of the key design decisions of FunnelWeb. In many cases, the alternatives have been unpleasant, but I am confident that in all cases, a fully workable solution has been found.

Chapter 2

FunnelWeb Implementation

2.1 Introduction

This chapter contains notes on the actual C code that implements FunnelWeb V3. This chapter is rather patchy. It has acted mainly as a dumping ground for ideas that I bothered to write about during development.

2.2 History of FunnelWeb Implementations

The first implementation of FunnelWeb (FunnelWeb V1) was written in Ada in December 1986. The project was initially canned as requiring too much time, but was resurrected when I decided to commit to Ada[**USDOD83**] and realized that I needed a program to write to help me to learn Ada. FunnelWeb V1 was, in fact, my first Ada program. It took about one month to write.

FunnelWeb V1 was used intensively by myself to write Ada programs from January 1986 to July 1989 at which time I finished my Ph.D. and lost access to the VAX. During this time at least twenty thousand lines of code were generated using FunnelWeb. Hardly anyone but myself used FunnelWeb.

After losing access to Ada and the Vax (and hence to FunnelWeb), I was forced back to programming the non-literate way. From time to time I found that I needed to use some of my old programs that I had written using Ada and FunnelWeb. I knew that Ada would become available on more machines, but certainly FunnelWeb wouldn't. I recognised a strong need for a portable version of FunnelWeb written in C but didn't have the time or energy to create one.

About this time (late 1989), David Hulse, at the time a student in Computer Science at the University of Adelaide, volunteered to translate the 4000 line Ada version of FunnelWeb into C. To my knowledge this translation process took about three weeks (in December 1989). The result was called FunnelWeb V2 and was formally signed into the public domain on 5 May 1992.

In general, David Hulse did a good job. However, the resultant code suffered from one or few serious defects, the most serious of which was a lack of portability.

Lack of portability of the C code, combined with the need for a rather solid design review, combined with the need to strengthen the program to bring it up to production standard, resulted in my performing a complete reworking of the code. The C code was entirely, but incrementally, replaced or reformatted. The code was also strengthened and new features were added. This process took about two months (November and December 1991). A further two months (approx) were spent writing documentation, constructing a regression test suite, porting the program to different machines, and sorting out the legal issues involved in its release.

I would like to take this opportunity to record a debt of gratitude to David Hulse who translated FunnelWeb from Ada to C. Although my reworking of his C code obliterated most of his code, his translation was pivotal to the development process. Without his effort in moving from Ada to C, I'm not sure that I would have mustered the energy and time to complete the process and bring FunnelWeb up to its current standard.

2.3 Why FunnelWeb Wasn't Used to Write Itself

After Knuth created the Web literate preprocessing system, he re-wrote it using Web and distributed the source code in Web source form. To allow the Web source code to be tangled by users not yet having a copy of Web, he also included the tangled Pascal code for the Tangler.

While this approach is heroic and serves to convey a commitment and a confidence in literate programming, it seemed to me that writing FunnelWeb in FunnelWeb would simply be asking for trouble. For a start, it would be very hard to modify any feature of FunnelWeb that had been used to write FunnelWeb, and the thought of what would happen if the working executable became inoperative for some reason does not bear thinking upon.

One million billion computer programs were written in the non-literate style before FunnelWeb was created. Why not one more?

2.4 Coding Style

Although FunnelWeb wasn't coded under any particular coding standard, it was coded in accordance with a fairly strict personal style of C which developed during the development of FunnelWeb. This style was subsequently embodied in a real C coding standard prepared for the South Australian Government Department of Lands.[1] Unfortunately, FunnelWeb was not formally developed under the standard and so some holes remain in FunnelWeb's coding style. This section aims to describe some of the more important aspects of the coding style.

Portability: This was a major goal of the FunnelWeb implementation. Two excellent books guided this move to portability. They were **[Rabinowitz90]** (which deals with C code itself) and **[Horton90]** (which deals with the portability of various library calls). Other works such as **[Kernighan88]** and **[ANSI]** were also helpful.

Identifiers: **[Rabinowitz90]**length specifies that for wide portability, identifiers of block and file scope should be unique to eight characters, and identifiers of program scope should be unique to six characters. I have gone further in FunnelWeb and actually made these restrictions actual limits on identifier length.

Because names must be so short, a system of abbreviations was developed to organize the identifiers used within FunnelWeb. Each abbreviation consists of a letter pair. Here are *some* of the abbreviations used:

```
bp - Body Part.
cm - Compare. Used to prefix comparison routines that return [-1,0,1].
dc - Document component.
dm - Dump package.
el - Element.
eq - Equal. Used to prefix comparison routines that return a boolean.
ex - Expression.
f  - Global files.
ll - List of lists.
ln - Line record.
ls - List Package.
lr - Lister package.
```

[1] The standard is currently unavailable, but is likely to be released or published eventually.

```
ma - Macro.
mc - Macro Call.
mn - Macro Name.
op - Options package.
pr - Parser.
ps - Position record.
sc - Scrap record.
sn - Section.
tb - Table package.
ty - Typesetter directive.
wf - Write file package.
wl - Write with EOL (misc.c).
wr - Write          (misc.c).
```

Pointers: Variables or types denoting pointers start with "`p_`".

Types: Names denoting types end in "`_t`". Thus, a type for a pointer to a table would be named `p_tb_t`.

File names: All files used in FunnelWeb have file names that are from one to eight characters long and file extensions that are from one to three characters long. This ensures that the files can be portably moved to all kinds of machines, even MSDOS!

2.5 Use of Memory

FunnelWeb is not a memory-stressed program. However, during its development, problems with the management of memory seemed to crop up again and again. This section documents some of these problems and the solutions adopted.

There are three places where memory can be obtained: the heap, the stack, and from static variables. The following three sections deal with each of these areas.

2.6 The Heap

One of the great frustrations of being a user is to find that a computer program is complaining about lack of memory when one knows full well that one has allocated at least ten times as much memory to the program as it would ever need to do its job. The reason for such error messages usually has to do with the programmer setting a fixed "reasonable" limit to a particular data structure and then locking it up into an array whose bound is specified by a constant. While the use of arrays can increase the speed of a program, it also means that the user cannot increase the capacity of the program without obtaining the source code and recompiling it, which is usually a daunting option.

The alternative is to use the heap for all data structures that can grow in proportion to the size of the user's input. This rule has been followed rigorously in FunnelWeb. This means that as memory spaces increase, users will be able to hand their version of FunnelWeb more memory without having to recompile it.

— § —

Some problems arose early on the Macintosh in the use of the heap. It seems that some of the allocations I was attempting to make were failing for some obscure reason, possibly my fault. Whatever it was, it went away when I replaced direct calls to `malloc` with calls to a mini package I wrote (called `memory`) that allocated large chunks of memory and then doled out small pieces as required by the rest of the program.

Having a package to manage all the memory allocation had two other benefits.

First, only one check was required in the entire program to see if memory had run out (in the memory package), and if that failed, the program could be brought to a screaming halt. This organization was far preferable to having each piece of code that needed to allocate memory having to check to see if `malloc` had failed.

Second, the decision to construct a mini-shell within FunnelWeb to support regression testing meant that FunnelWeb proper could be run many times in any given invocation of FunnelWeb. As a consequence it was necessary to make sure that there was no memory leakage between invocations of FunnelWeb proper. This was accomplished by reworking the memory package to operate a watermark system. The user of the package, when requesting memory, could request "temporary" or "permanent". If permanent, the memory package forgot that it had allocated the memory. If temporary, the memory package places the allocated block on a list. There was then a function in the memory package that could be called to deallocate all the temporary memory. Thus, so long as all requests for memory within FunnelWeb proper were for temporary memory, and that memory was freed at the end of every run, one could be sure that there was no memory leakage.

2.7 The Stack

For a while during the development of FunnelWeb a particularly nasty bug proved extremely hard to find. The symptom was that FunnelWeb would crash, sometimes at random, but more often upon entering a particular function. In the end about a day of specific debugging was required before the problem was tracked down to a stack problem. It turned out that somehow (either the fault of the Macintosh or the THINK C language system), only 6K was being allocated for stack space!!!!!!!

This experience led me immediately to go through the entire program and eliminate (or remove to the heap) any automatic variable declarations that used more than one hundred or so bytes.

The lesson is clearly that C programs that use more than a few thousand bytes of stack space are risking their portability. All large data structures should be placed in the heap.

2.8 Static Variables

Static variables also proved a problem on the Macintosh. It turns out that the Macintosh THINK C compiler does not allow more than 32K of statics *in the entire program*. For a while this restriction was a serious threat to the program as it was discovered that constant strings were included in this total! However, some searching revealed a compiler option that removed the strings from the static category.

Nevertheless, the 32K limit is rather severe. Again, it seems that for portability reasons, C programs that use a lot of static variables are risking their portability. As a result, the FunnelWeb code avoids static variables where possible in favour of the heap.

2.9 Implementing Text Indentation

At one point during the development of FunnelWeb, text indentation was fully implemented. However, it was subsequently removed because it was considered a dangerous feature. This section records the way in which text indentation was implemented so that if the feature ever has to be put back, this technique can be used again.

1. Create a new field in the `sc_t` record call `sc_postn`.

```
      char *sc_postn; /* Pointer in the range [sc_first,sc_last+1].           */
                      /* It is the minimum possible value of sc_postn for     */
                      /* which EOL does not appear in *sc_postn..*sc_last.    */
                      /* i.e. Points to the byte following the first EOL in   */
                      /* the scrap or sc_first if EOL does not appear.        */
```

2. Modify the scanner so that it generates this field. Sendtext should be modified so that it accepts an argument for the p_postn value.

```
LOCAL void sendtext P_((p_ps_t,char *,char *,char *,bool));
LOCAL void sendtext(p_tkps,p_first,p_last,p_postn,is_white)
/* Appends a text token to the end of the token list.                        */
/* IN: p_ps is a pointer to a position structure giving the position of the   */
/*     first character of the token.                                          */
/* IN: p_first and p_last point to the first and last byte of the text scrap. */
/* IN: p_postn has the same definition as sc_postn (see fwdata.h).            */
/* IN: is_white should be set to TRUE iff scrap is entirely whitespace.       */
p_ps_t p_tkps;
char  *p_first;
char  *p_last;
char  *p_postn;
bool   is_white;
{
 tk_t token;

 /* Empty text scraps should never be generated. */
 assert(p_first<=p_last,"sendtext: Text scrap bounds are bad.");

 /* If ch=EOL then we should be scanning more text, not shipping it! */
 assert(ch!=EOL,"senttext: Shipping text while still more to scan.");

 /* Check that p_postn is in range. See definition in fwdata.h. */
 assert(p_first<=p_postn && p_postn<=p_last+1,
        "sendtext: p_postn is out of range.");

 /* Debug: Check the p_postn field using a brute force check. */
 {
  char *i,*j;
  j=p_first;
  for (i=p_first;i<=p_last;i++)
     if (*i==EOL)
        j=i+1;
  assert(j==p_postn,"sendtext: sc_postn field is incorrect.");
 }

 /* Load the text token. */
 token.tk_kind        = TK_TEXT;
 ASSIGN(token.tk_ps,*p_tkps);
 token.tk_sc.sc_first = p_first;
 token.tk_sc.sc_last  = p_last;
 token.tk_sc.sc_postn = p_postn;
 token.tk_white       = is_white;
 token.tk_parno       = 0;
 ls_add(token_list,PV &token);
}
```

Then all the calls to sendtext have to be changed:

```
/* @ instructs FunnelWeb to replace the special construct with the */
/* special character. Luckily one appears just before the @ !!     */
/* Note: FALSE is OK because space is not a legal specialch.       */
/* Note: Setting parameter p_postn to p_ch-1 is OK as EOL is not a */
/*       legal specialch.                                          */
sendtext(&ps_spec,p_ch-1,p_ch-1,p_ch-1,FALSE);
break;

/* + instructs FunnelWeb to insert an EOL. We can't look to the end of */
/* the previous line to find an EOL as this might be the first line.   */
/* Running ahead to the end of the line is expensive, and having the   */
/* liner mini-package maintain a variable for it would be extra        */
/* housekeeping. Instead of all this, we just point to a static.       */
{CONST static char stateol = EOL;
 sendtext(&ps_spec,&stateol,&stateol,(&stateol)+1,TRUE);}
break;

/* If we hit something that ends a text token */
/* then we can transmit a white text token.   */
if (ch==specialch || ch==EOFCH)
   {sendtext(&ps_start,p_first,p_ch-1,MAX(p_sol,p_first),TRUE); return;}

/* Otherwise we have some more (non-white) text to scan. */
/* We can then send a non-white text token.              */
while (ch!=specialch && ch!=EOFCH)
   NEXTCH;
sendtext(&ps_start,p_first,p_ch-1,MAX(p_sol,p_first),FALSE);
```

The dump code needs to be changed too!

```
        wf_str(p_wf,"\"");
assert(token->tk_sc.sc_first !=NULL,"dm_tkls: NULL ptr1.");
assert(token->tk_sc.sc_last  !=NULL,"dm_tkls: NULL ptr2.");
for (i=token->tk_sc.sc_first; i<=token->tk_sc.sc_last; i++)
  {
   if (i==token->tk_sc.sc_postn)
      wf_str(p_wf,"<postn>");
   if (*i=='\n')
      wf_wl(p_wf,"");
   else
      dm_byte(p_wf,*((ubyte_ *) i));
  }
if (i==token->tk_sc.sc_postn)
   wf_str(p_wf,"<postn>");
wf_str(p_wf,"\"");
}
```

3. Over in the Tangle module, create a massive array of pointers to scraps to be used as a stack. Maintain pointers into the stack called `current` and *base* (similar to the blank indentation variables). Implement the following:

- To write out a scrap, scan it byte by byte. Output each byte. When you hit an EOL, pop the stack back to `base`. Then write out an EOL followed by the stack contents but writing each scrap only from `postn` to end end of each scrap. When you have finished the new scrap, push it on the stack.
- When you hit a new macro to expand, save `base`. Restore it later.

The `postn` field solves the big problem of how to cope with something like this:

```
The rain in Spain
falls mainly @<on the plain@>
```

The trouble is that we want to text indent the lines in `@<on the plain@>` with just "`falls mainly `". However, this string is only part of a scrap. The solution is to get the scanner to record, in the `postn` field of each scrap, the position of the first byte with a EOL-free run to the end of the scrap.

This scheme is very efficient because all we are doing is pushing and popping pointers to scraps on a stack array. The main disadvantage is that the array must necessarily be finite and would impose a limit on the depth of indentation nesting (not a big problem).

Chapter 3

FunnelWeb Modification

3.1 Introduction

This chapter deals with modifications to FunnelWeb. Although the GNU license under which FunnelWeb is distributed allows anyone to modify FunnelWeb and distribute the modified versions, there are a number of dangers in doing this. This chapter explains the dangers and presents some issues that you should think about if you intend to distribute modified versions of FunnelWeb.

3.2 The Danger of Modifying Languages

Whenever someone modifies a computer program they should always ask themselves "what is the impact of these changes going to be on users". For mature products, the degree of impact is largely determined by the investment that users have made in particular file formats supported by the product.

At the two extremes, there are the video game and the compiler. If one changes a video game, the change affects only the future games played. However, if one changes a compiler in particular ways, it is likely that thousands of users will be forced to change millions of source files to accommodate the new version. Thus the impact of changes to a program depends largely on the investment made by users, in source files, training, and even muscle memory!

FunnelWeb lies very much at the compiler end of this spectrum. In fact it lies even further. Not only does it implement a language, but it is currently the *only* implementation of that language! This means that, although they shouldn't, people will be tempted to treat the FunnelWeb program as the definition of its input language instead of the definition chapter in the *FunnelWeb User's Manual*, which takes precedence.

Because FunnelWeb defines an input language, and people will write programs using that language, particular kinds of changes in the language will impact severely on users. Three different kinds of change are discussed below.

> **Removing features:** Removal of features (featurectomy) is extremely difficult once the user base has started using them. If a feature is removed, users of the program will have to go through all their programs and find a way to simulate the effect of the removed feature with other features. Often this is so unthinkable, that bad language constructs are tolerated far beyond their useful lifetime.
>
> **Modifying features:** Modification of features has less direct impact than the removal of features, but can cause more subtle errors. For example, consider the sorts of subtle errors that might arise if the semantics of the FunnelWeb comment `@!` were changed so that it no longer includes the end of line marker. Anyone modifying features should be sure that they are not inadvertently laying traps.

Adding features: Although the addition of features is generally the most painless for the user community, as Hoare points out, it is also the most dangerous in the long run.

> "When any new language design project is nearing completion, there is always a mad rush to get new features added before standardization. The rush is mad indeed, because it leads into a trap from which there is no escape. A feature which is omitted can always be added later, when its design and its implications are well understood. A feature which is included before it is fully understood can never be removed later." **[Hoare80]**

These problems exist even when there is a central authority. If we consider the case where there is no central authority, and users modify a language and its implementations and distribute the modified versions, the result is even worse. For a start, the structure of the genealogy of the program changes from a list to a tree. This makes it impossible to impose an ordering on the different versions and makes it difficult to merge them once they diverge. Second, it reduces the portability of files created under different versions. Third, it makes adding features hazardous. If a feature is added in one version, files will be created that use it. These files will immediately become non-portable. Furthermore, two programmers may introduce different features that use the same syntactical constructs, thus preventing the two versions from ever being merged. Worse, two programmers could introduce *similar* features that use the same syntactical constructs, producing even more subtle problems.

The only way to avoid all these problems is to create some sort of central design authority, (or, at the very least, some sort of design synchronization) that controls the language and its implementation.

The benefits of tight control over the language are enormous.

Universal portability: Source files can be treated as portable. If all the language implementations in the world are singing the same tune, then someone in Sydney could send a file to someone in New York and be sure that it will be successfully processed.

Clear semantics: Doubt about the semantics of the language will be greatly minimized. So long as more than one slightly differing version of a language exists, there will be confusion over the semantics of its constructs. A good example is the confusion between the AT&T Unix and Berkeley Unix shell languages.

While changes to the language supported by a piece of software is the most serious problem associated with multiple versions of software, the lesser problem of keeping track of changes that don't affect the language that the software implements can also cause trouble. Merging different versions of software is extremely tedious as anyone who has tried it will verify. However, changes not affecting the language are far less serious because they impact only on the program itself, not on the far more numerous source files that depend on it.

3.3 Authority vs User Security

There are a number of ways of providing the strong central design authority required to produce the portability and semantic security desired by users,

- Trade mark the name of the language. Publish a specification of the language under the trade name. Warn all users not to trust any implementation that does not guarantee that it implements the language. Then control implementations by only licensing the trade mark to conforming implementations.
- Create a single implementation of the language. Do not release the source code to the implementation. Release only binary executables.
- Release the source code to the implementation, but under a license that prohibits the distribution of modified versions.

Many other variations on these themes are possible, but they are all based on the idea of regulating either the "official" definition of the language or all of its existing implementations.

At about this stage, a conflict arises. While users want the language they are using to be tightly controlled, they also want to feel secure about the availability of its implementations. If hundreds of source files are to be created in the language, there had better be one or more translators available when the time comes to use them.

In the case of widely-used programming languages such as C, the problem of securing implementations does not arise. If you don't get your compiler from Borland, you can get it from Microsoft. However, in the case of a newly introduced language such as FunnelWeb, there may be very few implementations, or even just one implementation, and this poses a danger to the user who is committing to that format. Despite the best intentions of the copyright owner, there are a variety of ways in which a computer program can suddenly become unavailable. Here are some examples:

- If the program is owned by a company and the company goes broke, the liquidators may not bother to place the software in the public domain.
- If the program is owned by a kindly individual who issues licenses for free, and that individual suddenly dies, users might suddenly find themselves negotiating with hard-nosed estate executors for the rights to continue using the program.
- If whoever owns the program does not upgrade the program every so often, it is actually possible for the released executables to "expire" when new operating system upgrades appear.

These possibilities mean that users about to invest in a newly introduced programming tool such as FunnelWeb have legitimate concerns when they worry about the tool's availability. For non-commercially distributed tools such as FunnelWeb, the simplest solution to this problem is to make the source code available, either by placing it into the public domain, or by releasing it under some kind of public license, such as a GNU license. The trouble with releasing the source code is that it removes control from the design authority. Once the code is "out" there is no stopping anyone from modifying the program and distributing modified versions. All chaos can ensue.

One solution to this conflict is to take the trade mark approach. This separates the right of the design authority to control the language and its implementations, and the right of the users to use the source code in any way they like (but they may have to change its name!). However, the trade mark approach is expensive.

The solution that I (Ross Williams (`ross@spam.adelaide.edu.au`)) have adopted is to release the FunnelWeb source code under a GNU license and then to write this chapter in the *FunnelWeb Hacker's Manual* to dissuade possible hackers from releasing modified versions under the FunnelWeb name. This is about as close to the GNU/trademark approach I can get without forking out about $1200 for a US trademark or paying even more to get a lawyer to attempt to draft a contract stating "thou shalt not modify the FunnelWeb language".

3.4 What I Want to Protect

The concerns expressed in the previous section about modifications to the FunnelWeb program do not preclude modifications. They merely imply that some conditions be observed when modifications are made. In fact, having formally released the sources under GNU, there is nothing that I can do to stop people distributing modified versions and the conditions to be described must be imposed voluntarily.

In the end there are two things that I want to protect/maintain:

1. Restriction of the name "FunnelWeb" only to computer programs that exactly implement my "official" definition of the language.

2. Restriction of the FunnelWeb source file extensions "`.fw`" (input files) and "`.fwi`" (include files) only to source files that conform to my "official" definition of the language.

So long as these conditions hold, source files created under "FunnelWeb" will be portable, and the language will have the potential to be stable and well-defined. Here are my suggestions for how to obey these rules. These suggestions are in addition to the GNU license rules about documenting any changes that you make in the source files.

> **Modifications that do not affect functionality:** If you change the FunnelWeb program in a manner that does not affect the functionality of the program in any way (e.g. port it to a new machine, or just speed it up), then you should modify the program to write out a message when it starts up saying that it is a modified version of FunnelWeb. No other actions need be taken.
>
> **Modifications that affect functionality:** If you make changes to FunnelWeb that affect its functionality (e.g. changes to command line options, the command language, or the input language) you should change the name of the program so that the name no longer contains the word "FunnelWeb", and should choose alternative input and include-file file extensions (the current ones are "`.fw`" and "`.fwi`"). For example, you might call your program "BananaWeb" and use the file extensions "`.bw`" and "`.bwi`".

These rules are not very restrictive. Basically you can do what you like so long as you change the name of the resulting program. I do not wish to restrict anyone who might want to use FunnelWeb as a foundation for a more sophisticated literate programming system. My sole aim here is to protect the integrity of what already exists.

3.5 Modifying the Manuals

While it is permitted to modify the FunnelWeb program, no license has been granted to modify its documentation (*The FunnelWeb User's Manual* and *The FunnelWeb Hacker's Manual*). The only operation that can be performed on the FunnelWeb documentation is a verbatim copy in any medium.

If you have created a modified version of FunnelWeb and wish to document it, either write your own manual from scratch, or write a manual that can be used in conjunction with the existing manuals.

3.6 How Copyright Law Works

In order to understand the next section, some understanding of copyright law is required. The assertions made in this section are not legal opinions; they are merely approximations of the law, based on my understanding of it.

By default, whoever creates a **work** owns **copyright** over it. This means that by default, that person is the only person in the world who has the legal right to make copies of the work.

Copyright over a work usually covers all direct derivations of this work. For example, if someone owns the source code to a computer program then they also own any direct derivations such as the binary executable.

More than one person may own copyright over a work. If this is the case, then the work cannot be copied without the permission of all of the people owning the copyright.

A person (or group) owning copyright over a work can **license** the right to make copies of the work to anyone he likes under any conditions he likes.

If a work is modified, the result is called a **derived work**.

If copyright on a work is owned by A is the work is modified by B, copyright on the derived work is owned by both A and B.

Copyright is not ownership over particular configurations of information, but rather is the right to restrict the copying of information derived from particular sources. For example, if two people point their cameras out a window and take identical photographs, each person has the right to restrict the copying of *their* photograph regardless of the fact that the other person has copyright over an identical photograph. In the unlikely event that two people wrote novels that were letter for letter identical and it could be proven that there was no contact between the two people when they wrote the novels, then copyright law would each protect the right of each person to restrict the copying of the novel originating from *them*. In practice, the probability of such an event occurring is so low that the law assumes that if two works are close or identical, that there has been some information interchange at some point.

Thus, if I own copyright on computer program X, and I create an identical copy of X called Y and distribute it under the terms of a license agreement, then that license agreement does not cover the copy X. Once the license agreement is signed, there is nothing I can do to undo the license; Y has gone. However, I still have total control over X.

Version 2 of the GNU General Public License (under which FunnelWeb is released) specifies (in a nutshell) that software can be copied freely so long as it is not sold, and that it can be modified and copied freely so long as the modifications are logged and it is not sold. In both cases, it requires that the result of the copy or modification operation be copied only under the conditions of the license.

Once a declaration is issued licensing a copy of a program under GNU license, that license cannot be retracted.

3.7 Management of the Official FunnelWeb

I intend to maintain an official copy of FunnelWeb and release it under GNU license from time to time so that everyone can use it. Unfortunately its going to be a little messy.

For a variety of reasons, I want to maintain total control over *my* copy of the official version of FunnelWeb. Some of the reasons are:

- If it turns out that there is an enormous demand for FunnelWeb (unlikely) then I want to be able to create a business based on it, or at least be able to bundle it with other products. This would be difficult if the only up-to-date copy available was a GNU copy. Note: If this happens, all the GNU releases of FunnelWeb will remain valid.
- I want to be able to copy code from my version of FunnelWeb into commercial products without having to worry about the GNU license.

The trouble is that (in my understanding of copyright law), once a copy of FunnelWeb has been released under GNU license, then it and any works derived from it fall under GNU license too. This means that if you modify a GNU release of FunnelWeb and send me your modifications, and I incorporate them into my version, then I no longer own copyright on the version I modified. Basically, if GNU touches a work, then that copy is gone forever.

My solution to the problem is to keep a copy of FunnelWeb that is GNU-clean and only make modifications to it that are GNU-clean. So long as my official copy is GNU clean, any modifications that I make myself will be GNU clean. The problem comes when someone else modifies the GNU version and then wants me to incorporate the modifications into my official version. As I understand the law, by default, the programmer owns the modifications (because he made them), but the result of making the modifications falls under GNU license. What I propose is that those who want their modifications included in the official copy of FunnelWeb sign an agreement transferring copyright of the modifications over to me on condition that the result be released under GNU license within a certain time period. **Figure 1** demonstrates the process.

There are two main disadvantages to this scheme:

```
+---------->----------+-------------<------------+
|(modifications by    |                          |
| me)                 V                          |
|     +----------------------------------+       |
+--<--|  My Official Copy of FunnelWeb   |       |
      |   Copyright (C) Ross Williams    |       |
      +----------------------------------+       |
                      | (Periodic releases)     / \
                      V                        /   \  Programmers
           +------------------+               /Legal\ sign away
           | GNU Release Copy |               \Filte/ copyright on
           +------------------+                \ r /  modifications
                      |                         \ /
                      | (Modifications made by   |
                      V  random programmers) --->--+
          +------------------------+
          | Modified GNU Version   |
          +------------------------+
```

Once a work is released under GNU license, all derived works fall under GNU license. However, this diagram shows how copyright can be maintained over a work, while still making it available under GNU license. Unfortunately, this scheme requires that copyright on modifications to FunnelWeb be signed off by the people who make the modifications.

Figure 1: How I am going to maintain copyright over FunnelWeb.

1. Getting each person who contributes modifications to FunnelWeb to sign and send a piece of paper signing away copyright will be tedious.
2. Programmers may be reluctant to sign away copyright on modifications because the result could be used by me for commercial purposes by me at a later date (Note: But it must be released under GNU as well).

These disadvantages are regrettable consequences of my decision to retain total control over the "official" version of FunnelWeb.

The upshot of all this is that:

- I will never allow any code from a GNU release of FunnelWeb to touch my official copy. Only modifications will be acceptable.
- If you make a modification to FunnelWeb and send it to me for inclusion in the official version, I will not incorporate your modification until I have received a signed declaration transferring copyright. I will prepare such a contract when the first person sends me a worthy modification!
- If you intend to make changes to FunnelWeb that are so massive that they will involve wide-scale modification of the source files, and you want the modification to be part of the official version, it might be a good idea to contact me before doing so. There are two reasons for doing this. First, it allows me a chance to comment on your proposed modifications and give my opinion on whether they are likely to be incorporated into the official version. Second, I can send you a copy (under special license) of the GNU-clean official version to work on. This will allow you to ship back your changes as files rather than as change sections.

As a separate point from the legal issues, I am likely to turn out to be pretty picky about what modifications I will incorporate into the official copy. Whether a modification will be accepted will depend, amongst other things, on the following criteria.

- Does the modification fit in with the design goals of FunnelWeb, or is it just a case of creeping featurism?
- How well coded is the modification? Would it reduce the quality of the code?
- If the modification changes FunnelWeb's functionality, is this a desirable change? How will it impact on existing users?
- Would it be quicker for me to make the modification myself than to work out how to incorporate the submitted modification?

It is my goal to guard the integrity of the design and code of the official version of FunnelWeb and so I will probably be rather fussy about what I regard as a worthwhile modification. I do not wish to lose control of the code, either technically or legally.

3.8 A GNU Version?

The controls on the official version of FunnelWeb described in the previous section will have at most a minor impact on FunnelWeb development if FunnelWeb already serves the needs of most of its users. However, if it turns out that FunnelWeb is both popular, *and* needs widespread modifications and enhancements, then the official feedback loop described in the previous section will become unworkable, and there will be a GNU breakout.

A **GNU breakout** will occur if FunnelWeb users become organized enough and annoyed enough (at the official development cycle) to start maintaining a coherent version of FunnelWeb that diverges from the official version. If a GNU breakout occurs, I would appreciate, as requested earlier, that a different name and file extensions be chosen for the diverging version.

3.9 Summary

Decentralized modifications to a programming language and its implementations can seriously damage the language's portability and semantic clarity. Unfortunately, there is a tension between centralizing control of a programming language and providing source code security to nervous users. In FunnelWeb, this trade-off has been resolved to some extent by releasing the source code along with a request that if the functionality of the program is modified, that the program's name be changed and a different file extension be chosen for input files created under the modified program. The author of FunnelWeb has decided to maintain an official version of FunnelWeb over which he will own copyright, but which he will release under GNU license from time to time. This means that copyright on all changes to FunnelWeb must be signed over to the author before they have a chance of being incorporated in the official version. Finally, it is possible that this reluctance of the author to remove his copyrighted version from the development cycle of the official version will result in a GNU breakout in which a diverging GNU version of FunnelWeb will be maintained by the GNU community. The author hopes this won't be necessary and requests, if this happens, that the diverging GNU version be renamed.

Chapter 4

FunnelWeb Future

Previous chapters have explained the design of FunnelWeb and given reasons for why it is the way it is. This chapter goes through each of the aspects of FunnelWeb, and explores their problems, how they can be solved, and lists opportunities for improvement.

4.1 Documentation

An official example: A official example program written using FunnelWeb should be constructed and made available.

Index program: In order to typeset the documentation, a portable index sorting program is required. One should be written and added to the distribution kit. Perhaps this could be the official example!

4.2 Command Line Interface

Buffer length: Currently the FunnelWeb shell uses the `COMLINE_MAX` constant to size its internal command line buffers. This is untenable. The maximum length of a shell command line should not be machine dependent.

Antiquated Features: As the FunnelWeb language develops, it is likely that some changes will have to be made that will render one or more language constructs obsolete. When this happens, it may be necessary to add a command line option that has the power to turn on and off warnings or errors flagging antiquated features.

4.3 Shell Interpreter

SETALL command: When writing FunnelWeb scripts, it is sometimes desirable to set *all* of FunnelWeb's options to some particular value so that the script is not vulnerable to changes in FunnelWeb's default values which might occur from time to time. To this end, it may be worth creating a "`SETALL`" command that is identical to the "`SET`" command except that it will generate an error if the value of an option is not specified explicitly.

Recursion test: A test should be added to test for recursive invocation in a shellscript.

Diagnostic counting: The code for counting diagnostics in the script interpreter is rather messy and perhaps even buggy. It should be cleaned up and commented.

Argument counting: In the `command.c` module, there is a variable that counts the arguments to a command. Currently it takes the value of the number of parameters including the command verb. This has turned out to make the code less readable. It should be changed to be the number of arguments to the command verb.

Make facility: It may be worth building some sort of make facility into the script language so as to support machines such as the Macintosh that do not already have this facility.

Signature file: One problem with using FunnelWeb in conjunction with an external `Make` facility is that a user might change a FunnelWeb source file without making changes that will affect the files that it generates. If FunnelWeb is then run and the "`+D`" option is on, then the output files will be deleted (to avoid further `Make` propagations). If `Make` then has a production linking the `.fw` file to its output files, then it may conclude that the output files are still out of date. To solve the problem, FunnelWeb could be changed to write a `.sig` file whenever it processes a `.fw` file. The `Make` production could then be wired up to link the `.fw` file to the `.sig` file instead of to the output files.

4.4 Language Design

Some proposed changes to FunnelWeb do not correspond to any particular component of FunnelWeb and are really to do with the design of the input language.

Output or file?: The "`@O`" special sequence for defining an output file is somewhat non-mnemonic and can be confused with zero ("`0`"). Perhaps it should be replaced by the "`@F`" sequence.

Syntax of section names: Currently section names use the same syntax as macro names. For example "`@<Sloth@>`". It can be argued that angle brackets should be reserved only for macro names and that some other syntax should be found for delimiting section names. This is not a clear issue. It could also be argued that they are both names, and that because sections can inherit their names from the macros they contain, that the names are of the same "type".

One macro per section: One particular style of using FunnelWeb is to have at most one macro definition per section. It may be worth adding a pragma that instructs FunnelWeb to enforce this.

Should @{ suppress EOL?: When defining a macro in FunnelWeb, it seems to be rule rather than the exception that the "`@{`" be followed by "`@-`". This suppresses the EOL on the definition line, allowing the first line of the macro to be placed immediately above and in line with the other lines without introducing an EOL at the start of the macro text. One option is to introduce a pragma to determine whether to suppress EOLs following "`@{`".

Pragma syntax: It is not clear how "loose" the syntax of pragmas should be. Perhaps they should be case insensitive.

Conditionals: Depending on demand, it may be worth reintroducing some sort of conditional feature into FunnelWeb. However, it would have to be very simple to compete with the various ways in which conditionals can already be fudged within FunnelWeb as it stands.

File markers: It might be worth modifying the language so that a special syntactical marker is required at the start and end of files. This will assist in detecting truncations and other corruptions.

Formal parameter lists: It might be worth changing over to a syntax for formal parameter lists that does not require the `@(` and `@)`. However, they could be retained as optional for backward compatibility.

4.5 Scanner/Mapper

All non-contiguous mappings: Currently FunnelWeb requires that all input files be mapped into a contiguous lump of memory. This caused problems for two reasons. First, to do this, one has to allocate the memory first, and to do that, you have to know how long the file is, and it turns out that finding out the length of a file in a portable manner is very inefficient. Second, although

IBM PC compatibles may have megabytes of memory, it is segmented into blocks of at most 64K. This means that FunnelWeb currently cannot read a file longer than 64K on a PC.

These problems could be avoided if the mapper and scanner were reorganized to allow input files to be read in and stored as a linked list of chunks of text rather than a contiguous block.

EOL is unspecifiable: FunnelWeb uses ASCII character decimal ten (10) internally to represent logical end-of-line and is currently organized so that if one of these is inserted into the text by the user using a "`@^D(10)`", it will be written out as a logical end of line, rather than as a single ASCII character 10. This should be fixed.

Allow mnemonics for unprintables: FunnelWeb allows users to insert unprintable characters into the output using the "`@^D(ddd)`" special sequence. Perhaps it would be changed so that it understands ASCII standard mnemonics such as "`LF`" as well as ASCII numbers.

Version pragma: A "`version`" pragma should be added that allows the user to specify in the input file the version of FunnelWeb that was around when the input file was created. At a later date, such a construct would be very useful for determining how an input file should be updated if the FunnelWeb language has changed between versions.

4.6 Parser

There are no proposals to change the parser except as a consequence of other proposals.

4.7 Analyser

Recursion detection: Currently the FunnelWeb analyser flags, with an error, all macros with an infinite expansion. This would be best changed to flagging all macros that are directly or indirectly recursive. To do this, Tarjan's algorithm[**Tarjan72**] for the detection of strong components should be installed.

Once only macros: By default FunnelWeb prevents a macro from being called more than once unless it has a "`@M`" associated with it. However, FunnelWeb does allow a macro that calls such a macro to be called more than once. Perhaps this "loophole" should be plugged somehow.

4.8 Tangle

The Tangler is one of the cleanest components of FunnelWeb, as basically all it has to do is expand some very well-defined macros.

Text indentation: Currently FunnelWeb supports *no indentation* and *blank indentation*. A third form could be added if it was considered necessary. *Text indentation* is the same as *blank indentation* except that instead of indenting with blanks, FunnelWeb would indent with the text to the left of the called macro. This facility could be useful for commenting out large bodies of text in languages that do not have multi-line comments (e.g. Ada). A discussion of the pros and cons of this form of indentation appears earlier.

4.9 Weave

Perhaps FunnelWeb's weakest aspect is its typesetting facility.

Align table of contents: When FunnelWeb generates a table of contents, the section numbers are not quite aligned with the start of the controlling heading above them.

Typesetting strength: It should be possible to specify the level of typesetting strength for headings so that short documents do not look overdone when typeset. A new pragma would be good for this.

Typeset a portion: Sometimes it is desirable to typeset just a portion of a program. A command line option could be added to do this. The option could accept as its argument, a string containing a list of section numbers or heading names.

Generic typesetter option: In addition to building in a number of different versions of Weave, one for each popular typesetter, it would be possible to add a special generic format where the typeset output is expressed in terms of *FunnelWeb macros*. The user could then specify macro definitions for a non-supported typesetter and run the output through FunnelWeb Tangle to get a typeset file in a format suitable for the unsupported typesetter.

Suppression of include files: It should be possible to specify in the input file that particular included files not appear in the typeset output. Currently, the fact that an inclusion has occurred is not even represented in the typeset output. Suppression of inclusions is particularly necessary where a library of macros has been included at the top of each of a group of source files.

Cross reference of identifiers: WEB provides a list of identifiers and a list of all the definitions in which they are used. A similar feature could be added to FunnelWeb.

Support for non-printables: Currently FunnelWeb does not provide support for typesetting the special "`@^(num)`" sequences. This should be added.

Support for @+ sequences: Currently Weave does not see "`@+`" sequences as such. Instead it perceives them as ordinary EOLs. However, there are arguments for typesetting them specially.

Typeset text in macro bodies: One of the much-loved features of WEB is the way that it allows the user to switch recursively between document and program formats. FunnelWeb does not allow this, but should. In FunnelWeb, the delimiters "`@{`" and "`@}`" are already used consistently to delimit macro text. The "`@[`" and "`@]`" sequences have been reserved for the delimitation of documentation text.

Non-tt typesetting: The current version of FunnelWeb sets all its macro text in `tt font`. This is both a blessing and a curse. It is a blessing because it connects the reader directly to the code, with no complicated intermediary. It is a curse because it looks ugly compared to the beautifully typeset programs produced by other literate programming tools.

The difficulty with adding such beautiful typesetting is that it is necessarily language-specific. Keywords and syntax differ from language to language and it would not be easy to come up with a set of language independent rules.

One approach is to write a set of Weave back-ends, one for each language. Another approach is to *generate* back ends. This is the approach taken in the *Spider* system**[Ramsey89]**. In the *Spider* system, the programmer writes production rules for converting lexical components of the program text into typesetter instructions. The *Spider* program reads these rules and generates a new version of WEB suited for the target language.

For FunnelWeb a slightly different system is proposed in which Spider-like rules appear in the input file and are used directly by Weave to perform the typesetting. An intermediate abstract typesetting language could be used so that the productions can be made language specific, but not typesetter specific.

4.10 Lister

Glue factor: A glue factor could be added that determines how many lines can be in between two diagnostics in the listing before the two groups of lines are joined together in the listing with no intervening ellipsis.

4.11 Diagnostics

Advisory information: Some of FunnelWeb's diagnostics provide a detailed explanatory paragraph. While this information might be useful the first time, it has the capacity to clog up a listing file if the user has made the same error many times. To solve this problem, FunnelWeb could be modified so that such explanations are only displayed the first time the error occurs.

Abort after n errors: A facility could be added to prevent FunnelWeb's scanning, parsing, and analysing phases from continuing if a certain number of errors have already been issued.

4.12 Speed

Measurement of speed: Although FunnelWeb can generate a breakdown of where it is spending its time, it does not give a final rating in lines per minute. This should be added.

Find the hot spots: Although FunnelWeb has been designed to allow high speed, not much effort has so far been made to make it fast. This should be done.

Change some declarations: FunnelWeb is full of variable declarations where the variables are wider than they need be. Replacing these might speed it up.

4.13 Correctness

Bounds analysis: Not much effort has gone into the design of FunnelWeb's input boundaries. An analysis should be made of FunnelWeb's behaviour when the following quantities are stretched:

- Input line length.
- Input file size.
- Number of macros.
- Length of macro.

In particular, FunnelWeb's behaviour at 32K and 64K boundaries should be observed.

Stack detection: Macintosh THINK-C provides just 6K of memory for the stack. It might be worth adding checks to make sure that the stack is not being blown.

4.14 Test Suite

The following tests should be added to the test suite:

```
Lister
------
   LR01: Test with a full listing with no diagnostics.
   LR02: Test with a full listing with diagnostics.
   LR03: Test with an abbreviated listing with no diagnostics.
   LR04: Test with an abbreviated listing with diagnostics.
   LR05: Test error context system with nearby diagnostics.

Boundary Cases
--------------
Static analysis might preclude the need for most of these tests.
```

```
   BC01: Test what happens when memory runs out.
   BC02: Test on a file with a single line of a megabyte.
   BC03: Test on a file of a megabyte of EOLs.
   BC04: Generate an output file with an extremely long line.
   BC05: Generate an output file with one million lines.
   BC06: Test on a file with very many macros.

General
-------
   GN01: A large legal input file exercising as many features as possible.
         1. Test listing file.
         2. Test output files.
         3. Test typeset file.
   GN... A selection of ten real-life FunnelWeb files.
```

4.15 Machine-Specific Changes

Icon for the Macintosh: Currently no icon is supplied for the Macintosh version of FunnelWeb. An icon depicting a spider or a funnelled web of some kind would seem appropriate.

4.16 Summary

This chapter has describes some of the problems with FunnelWeb and some of the opportunities that exist for improving it. The direction and speed in which FunnelWeb will develop will depend largely on user feedback. If you have any strong feelings on where FunnelWeb should go, please email them to Ross Williams (`ross@spam.adelaide.edu.au`).

Appendix A

GNU General Public License Version 2

This appendix contains a verbatim copy of Version 2 of the GNU General Public License under which the FunnelWeb computer program is released. Note that the *FunnelWeb User's Manual* and *FunnelWeb Hacker's Manual* are released under a much simpler license that does not allow modifications.

```
    GNU GENERAL PUBLIC LICENSE
       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                          675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

    Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.
```

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this

License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
```

```
    type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

```
The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

Appendix B

References

[ANSI] Australian Standard AS 3955-1991, "Programming Languages — C", (ISBN: 0-7262-6970-0), 12 July 1991. Identical to: International Standard ISO/IEC 9899: 1990 Programming Languages — C.

[Bentley86] Bentley J., "Programming Pearls: Literate Programming", *Communications of the ACM*, Vol. 29, No. 5, pp. 364–369, 1986.

[Hoare80] Hoare C.A.R., "The Emperor's Old Clothes", Turing Award Lecture, 1980, from "ACM Turing Award Lectures: The First Twenty Years 1966–1985", ACM Press, New York, (ISBN: 0-201-07794-9), 1987 (see p. 155).

[Horton90] Horton M.R., "Portable C Software", Prentice Hall, (ISBN: 0-13-868050-7), 1990.

[Kernighan88] Kernighan B.W., Ritchie D.M., "The C Programming Language", (second edition,"ANSI C"), Prentice Hall, (ISBN: 0-13-110362-8), 1988.

[Knuth83] Knuth D.E., "The WEB System of Structured Documentation", (Web User Manual, Version 2.5, November, 1983), Stanford University, 1983.

[Knuth84] Knuth D.E., "The TEXbook", Addison-Wesley, (ISBN: 0-201-13448-9), 1984.

[Lamport86] Lamport L., "LATEX: A Document Preparation System", Addison-Wesley, (ISBN: 0-201-15790-X), 1986.

[Rabinowitz90] Rabinowitz H., Schaap C., "Portable C", Prentice Hall, (ISBN:0-13-685967-4) 1990.

[Ramsey89] Ramsey N., "Weaving a Language-Independent WEB", *Communications of the ACM*, Vol. 32, No. 9, pp. 1051–1055, 1989.

[Tarjan72] Tarjan R.E., "Depth-First Search and Linear Graph Algorithms", *SIAM Journal of Computing*, Vol. 1, No. 2, pp. 146–160, June 1972.

[USDOD83] "The Programming Language Ada Reference Manual", American National Standards Institute Inc, ANSI/MIL-STD-1815A-1983, 1983.

Index