crep:
a regular expression-matching textual corpus tool

USER'S MANUAL

Darrin Duford
Department of Computer Science
Columbia University
April 16, 1993

Technical Report
CUCS-005-93

TABLE OF CONTENTS

Attempt the end, and never stand to doubt;
Nothing is so hard, but search will find it out.

Robert Herrick, "Seek and Find"

1. INTRODUCTION

1.1 Definition and Purpose

crep[1] is a UNIX[2] tool which searches either a tagged or free textual corpus file and outputs each *sentence* that matches the specified regular expression provided by the user as a parameter. The expression consists of user-defined regular expressions of words and/or part-of-speech tags. The purpose of **crep** is to make the searches faster and easier than by either a) searching through corpora by hand; or b) constructing a lexical scanner for each specific search. **crep** achieves this facilitation by offering the user a simple expression syntax, from which it automatically constructs an appropriate scanner. The user therefore has the ability to execute a whole search in one command, invoking implicitly and explicitly several tools, including a sentence delimiter, a part of speech tagger (developed by Ken Church at AT&T Bell Laboratories), and various output filters.

1.2 Black-box Input and Output

Inputs:

1) The regular expression;
2) The corpus (either tagged or untagged);
3) a definitions file to alias complex expressions (optional);

[1] **crep** requires the UNIX kshell.

[2] UNIX is a registered trademark of AT&T Bell Laboratories.

4) various output formatting options (optional).

Outputs:

1) the sentence(s) which matched the specified expression;
2) the phrase(s) which matched in each sentence (optional).

1.3 When to use and not use crep

1.3.1 When to use crep

Anyone who wishes to search free text for regular expressions may find a use for **crep**. **crep** is especially useful when:

a) searching for certain parts of speech in a random or arbitrary order in a sentence;

b) mixing parts of speech with other regular expressions in the search (both kinds of expressions may can be articulated on the fly);

c) discovering sentences which include two or more arbitrary regular expressions separated by *n* words (even no words);

d) seeking all "short" or "long" sentences of size *n* in a given corpus;

e) searching for certain words used as certain parts of speech in certain places in the sentence, relative to the beginning of a sentence, the end of a sentence, or to another regular expression.

In general, **crep** executes searches, domain specific or otherwise, which are based on a corpus of words and sentences. The examples in the chapters to follow will paint a clearer picture as to what **crep** can do. The above list is not meant by any means to be exhaustive; the user is encouraged to discover other applications.

The output from **crep** may be useful to determine certain elements of domain-specific writing style, to discover new or unusual constructs of words, or to determine the prevalence of certain common or uncommon lexical constructs. This information can then eventually be utilized as rules for natural language generation engines.

1.3.2 When not to use crep

If one wishes to search for all occurrences of the variable `foo` in arbitrary C-code, `grep` should be used. Since **crep** returns whole sentences rather than physical line numbers of files, the user may see some pretty bizarre 'sentences' extracted from the code.[3] Furthermore, **crep** was designed with more complicated searches in mind; `grep` outperforms **crep** for a small search such as this one. Using **crep** for such a simple search would be like trying to cut down a blade of crabgrass with a chain saw.

1.4 Philosophy of crep

crep aims to be user-friendly in two primary ways:

a) abstracting the user from the nuts-and-bolts drudgery of compiling specific lexical scanners, whether differences are minor or major with respect to any existing custom-made scanners. This abstraction provides an environment cultivating what-if? questioning;

b) returning meaningful error messages whenever possible.

crep aims to be flexible in two ways:

a) frequently allowing the user to achieve the same end through two or more methods, according to user's preference and for the user's convenience;

b) giving the user many customization options, from altering the sentence delimiter to adjusting input and output formatting.

Throughout this manual the user shall see examples of **crep** which will hopefully illustrate to the user the above user-friendliness and flexibility.

1.5 What this manual assumes of the reader

1) Experience with basic UNIX commands (such as `cat` and `time`) and tools such as piping ('|') and redirection ('>' and '<'). Following are references on the UNIX operating system:

[3] **crep** does, however, let the user redefine what should be deemed the end of a sentence by giving the user two tools to build one's own sentence delimiter -- for example, one which delimits only at semi-colons. See Section 6 for a complete description of the tools **build_delim** and **build_delim_user**.

- Arthur, Lowell Jay. *UNIX Shell Programming.* New York: John Wiley & Sons, 1990.

- Bolsky, Morris I. and Korn, David G. *The Kornshell Command and Programming Language.* New Jersey: Prentice Hall, 1989.

- The UNIX man pages for `ksh`.

2) A strong knowledge of `lex` syntax for regular expressions. Examples of `lex` syntax are presented throughout this manual. Following are references on `lex`:

- Chapter 13 in *UNIX Shell Programming* (above).

- *Lex & Yacc.* Sebastopol, CA: O'reilly & Associates, Inc., 1992.

- The UNIX man pages for `lex` and `flex`.

1.6 Organization of this manual

Section 2 provides a basic tutorial on the use of **crep** and its expression syntax; examples abound. Section 3 introduces, by example, all other options of **crep** which can be given on the command line, with the exception of the custom sentence delimiter authoring options (Section 6). Section 4 reveals some information on **crep's** chain of execution which may aid in one's understanding of how **crep** functions. Several tips and suggestions to maximize usefulness of **crep** appear in Section 5; this section is a must-read. Section 6 provides a tutorial for the custom sentence-delimiter authoring facility of **crep**. Section 7 introduces several other tools which work with **crep**. Appendix A lists a summary of all **crep** options. Appendix B lists the physical location (in Columbia University's CS file system) of **crep** and the sample files used throughout the manual. Appendix C discusses some error navigation. Finally, a listing of the part-of-speech tags given by `pos`, the tagger used by **crep**, appears in Appendix D.

1.7 Conventions followed in this manual

Different typefacing has been used to represent different entities. Following is a summary of the conventions. Exceptions to these conventions are explicitly noted.

Sample	*Signification*
`$cat  sample`	user-given input at the command line and highlighting of selected portions of **crep's** output
`Expression`	output of **crep**, variable names, file names, and names of other UNIX tools
crep	how **crep** and the names of its associated tools appear in this manual; the first appearance of **crep**-specific terms
delimiter	In examples, clarification added which is not part neither input nor output; anywhere else, emphasis
Why?	rhetorical questions
usefulness	regular text of the body of the manual.

1.8 Acknowledgements

The author would like to thank Jacques Robin for editing the manual and extensively testing the software; Ken Church for the use of his tagging tool `pos`; and Judith Klavans for valuable information concerning the Brown Corpus tags.

2. USING crep WITH THE crep EXPRESSION SYNTAX

2.1 Introductory Example

The first example represents **crep** at its simplest. It uses a textfile `sample`, a toy collection of basketball article lead-ins, as the corpus[4]. Note that **crep** accepts input from standard input (by default), and outputs to standard output (by default). These defaults may be changed, as the user will see, in Section 3's examples.

There are several command line switches which can hold an expression as a parameter. Each call to **crep** requires exactly one of them. The most commonly used switch is `-e`, used here:

EXAMPLE 1.

```
$cat  sample
ATLANTA (UPI) -- Dell Curry scored 20 points
and Kelly Tripucka added 18 as Charlotte
took its third straight win in a 123-111
victory over the Atlanta Hawks.

BOSTON (UPI) -- Kevin McHale's 23 points
```

[4] the boldfacing in all examples has been added for clarity.

```
led six Boston players in double figures Friday
night as the Celtics defeated the Phoenix
Suns 132-103 for their 15th consecutive home
victory.

CHICAGO (UPI) -- Michael Jordan led a
balanced scoring attack with 24 points Wednesday
night and the Chicago Bulls coasted past the
Washington Bullets 118-94 for their third
straight win.

LANDOVER (UPI) Jeff Hornacek and Cedric
Ceballos scored 20 points each to lead six
Suns in double figures Friday night as
the Suns coasted past the Washington Bullets 117-91
for its fifth straight victory.

MINNEAPOLIS (UPI) -- Tom Chambers scored 28
points Sunday and the Phoenix Suns extended
their winning streak to six games with a 123-109
victory over the Minnesota Timberwolves.

$cat  sample  |  crep  -e  'Celtics'
Parsing Your Regular Expression(s)...
crep: Expression Parser Warning:
  Expression `Celtics` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: Celtics
Compiling the flex file...
Executing Flex File...

Matching sentences:

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory.

$
```

In this degenerate example, **crep** works like `grep`, except that the output returned is the whole matching sentence, not just the physical line in the corpus file. To expand the usefulness of **crep**, the **variable definition file** shall now be introduced.

2.1.1 The variable definition file

What is a variable definition file?

In keeping in line with the idea of simplifying searches, the user may define macros to represent arbitrarily complex regular expressions, and have access to these macros from the command line using the simple macro names the user assigns to them. These macros are edited in a separate file, and are made known to **crep** as a parameter of the `-d` or `-D` command line options. Variable definition files are especially useful for domain-specific expressions.

What does it look like?

A variable definition file specified to crep by the `-D` option looks like the rules section of a regular `lex` source file. Example 2 lists a sample variable definition file of this form. A variable definition file specified with `-d` contains definitions in **crep** rather than `lex` syntax. This option will be discussed after the **crep** expression syntax has been fully introduced (Section 2.3).

What does crep do with it?

By default, **crep** uses this file as it translates the user's expression on the command line into `lex` code, since **crep** uses `lex` code internally. During the conversion process, **crep** checks to see if the expression on the command line matches a name in the user's variable definition file. In example 1, for instance, **crep** could not find a variable definition by the name of 'Celtics', since no variable definition file was specified, so 'Celtics' was dropped into the resulting `lex` expression literally. In example 2, we shall use a variable definition file `ExpFile`, tailor-made for our corpus of basketball articles. The '@'s are markers which separate the actual word from a capital abbreviation which stands for a certain part of speech (Appendix D holds a complete listing of the tags).

A HELPFUL TIP

The user will see `flex` used in place of `lex` in various **crep** messages; `flex` is simply a GNU version of `lex`, called by **crep** for efficiency reasons. The input grammar is identical to that of `lex` grammar.

Example 2 will also introduce the `-w` option, which outputs the regular expression and the actual matching phrase below each matching sentence. A brief interpretation of one of the definitions appears after the example.

EXAMPLE 2.

```
$cat  ExpFile
SPACE               [ ]+

TEAM_NAME           ([tT]he@AT[ ][A-Z][a-z][a-zA-Z]+@NNS)|([tT]he@AT[ ][A-
Z][a-z][a-zA-Z]+@NP)|([tT]he@AT[ ][A-Z][a-z][a-zA-Z]+@NPNP[ ][A-Z][a-z][a-
zA-Z]+@NPNP)

SCORE               [0-9]+@CD{SPACE}[0-9]+@CD|[0-9]+@CD{SPACE}(to@[A-
Z]+){SPACE}[0-9]+@CD|[0-9]+-[0-9]+@CD

N_WIN               ((victory|win|triumph|decision|romp)@NN)|(win@VB)

STRAIGHT            (straight|consecutive)@

POSS                (my|your|her|his|its|our|their)@JJ

VBD_WIN             ((defeated|beat|downed|edged|pounded|routed|minced|
troubled|dumped|crushed|stopped|outlasted|upended|blasted)@)|(blew@[A-Z]+[
]out@)|(coasted@VB[A-Z][ ]past@)|(held@[A-Z]+[ ]off@)|(knocked@[A-Z]+off@)

ORD               ((first|second|third|fourth|fifth|sixth|seventh|eighth
|ninth|tenth)@CD)|([1-9][0-9]*((0|[4-9])th)|1st|2nd|3rd)@

VBG_ENABLE          (helping|pacing|guiding|lifting|sparking|igniting
|topping|powering|rallying|propelling|sending|carrying|engineering|spurring
|pushing|fueling|bringing)@VB
$
```

Take a look at `TEAM_NAME`, for example. This expression means (in English):

> *match the word 'the' (and its tag) and then either one or two proper nouns (and their respective tags).*

Thus, a `TEAM_NAME` would match, for instance, 'The Bulls' and 'The Boston Celtics' provided that the corpus is tagged properly. By properly we mean that the tagging process, occuring automatically and implicitly in the call to **crep**, tagged the words with the tags the user is looking for (for instance, the word 'my' in the definition `POSS` requires the tag `JJ`, the tag for an adjective, to match in the tagged corpus. Section 5.1 discusses the tagger, and Appendix D holds a listing of the tags for each part of speech). Now back to our example:

```
$cat  sample  |  crep  -e  'VBD_WIN'  -D  ExpFile  -w
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {VBD_WIN}
Compiling the flex file...
Executing Flex File...
```

```
Matching sentences:

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory.

   {VBD_WIN}: defeated@

----------------------------------------------------------------------------

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted  past the Washington
Bullets 118-94 for their third straight win.

   {VBD_WIN}: coasted@VBD past@

----------------------------------------------------------------------------

LANDOVER (UPI) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted  past the
Washington Bullets 117 91 for its fifth straight victory.

   {VBD_WIN}: coasted@VBN past@

---------------------------------------------------------------------------
```

The '@'s represent the first character of the tags which are implicitly appended to *every* word in the corpus. These tags are the bread and butter of **crep's** ability to match expressions containing part of speech information.

Notice here that **crep** found `VBD_WIN` in the file `ExpFile.` **crep** translated `VBD_WIN`, an expression in **crep** sytntax, into `lex` syntax, which is simply '`{VBD_WIN}`' for this example. The reader will begin to see that the **crep** syntax will differ (in the direction of simplicity) more and more from its `lex` counterpart as a function of the operation. The user is not, however, over a barrel -- `lex` syntax may be used instead of **crep** syntax, or even side by side with **crep** syntax. Section 3.1.1 covers the former option; 2.2.3.4 covers the latter.

> ***A HELPFUL TIP***
>
> Only letters, numbers, and the underbar character ('_') may be used macro names.

Note that **crep** echoes the `lex` expression it translates out of the expression specified by `-e` in the line '`Regular Expression: {VBD_WIN}`'[5]. This is the **crep** parser at work -- translating from **crep** syntax into `lex` syntax. A complete specification for **crep** expression syntax follows.

2.2 Expression operators

The **crep** expression syntax supports several ways of searching for subexpressions which make up a larger expression. By means of operators, **crep** can realize varied corpus-probing functionality.

There are five operators which always appear between two expressions. These operators are (in their surrounding context):

`exp1 #- exp2`	the two adjacent expressions `exp1` and `exp2` must be separated by *at most* # words and `exp1` *must* appear before `exp2` in the sentence
`exp1 #= exp2`	the two adjacent expressions must be separated by *exactly* # words and `exp1` *must* appear before `exp2` in the sentence
`exp1 #+ exp2`	the two adjacent expressions must be separated by *at least* # words and `exp1` *must* appear before `exp2` in the sentence
`exp1 . exp2`	`exp1` must appear before `exp2` in the sentence. The number of words in between the expressions is limited only to the boundary of the sentence.
`exp1 ; exp2`	`exp1` and `exp2` must both appear in the sentence but *in any order*

Note: omitting the in-between operators (for instance, searching for '`exp1 exp2`' with no operator in the middle) will produce incorrect output because **crep** will not know how to search for `exp1` and `exp2`.

[5] and, in turn, this `lex` macro definition is shorthand for '`((defeated|beat|downed|edged|pounded|routed|minced|troubled|dumped|crushed|stopped|outlasted|upended|blasted)@)|(blew@[A-Z]+[ ]out@)|(coasted@VB[A-Z][ ]past@)|(held@[A-Z]+[ ]off@)|(knocked@[A-Z]+ off@)`', as specified in the file `ExpFile`.

2.2.1 Special Operators

There are five special operators which do not occur *between* expressions.

These two occur *around* expressions:

`(exp1|exp2)`	either `exp1` or `exp2` must appear in the sentence
`@@@...@@@`	treats *everything* between the '`@@@`'s literally; **crep** does not parse anything between '`@@@`'s. Straight `lex` syntax must be used here.

These two occur *in place of* expressions:

`@BEG@`	Signifies the beginning of a line. It means nothing by itself; it is used to find expressions appearing an arbitrary word distance from the beginning of the sentence.
`@END@`	Signifies the end of a line. It is analogous to `@BEG@`. It is used to find expressions appearing an arbitrary word distance from the end of the sentence.

The '`?`' operator appears *after* an expression:

`(exp)?`	Signifies that `exp` is optional. The optional expression must be surrounded by parenthesis.

2.2.2 Examples of some operator combinations

`exp1 ; exp2 ; exp3`	`exp1`, `exp2`, and `exp3` must all appear in the sentence but in any order
`exp1 2- exp2 ; exp3`	the superexpression '`exp1 2- exp2`' can appear either before or after the expression `exp3`
`@BEG@ 2- exp1`	the expression `exp1` can have at most two words appear before it in the sentence

A HELPFUL TIP

All operators must be separated by at least one space from the surrounding expressions except (, |,), ?, and `@@@`.

Most characters which make up the operators can be typed without holding down the shift key. This choice was intentional.

2.2.3 crep expression operator examples

2.2.3.1 '#-' and '#='

The next example makes use of the '#-' and '#=' operators. '#-' specifies that the surrounding expressions must be separated by *at most* '#' words. '#=' specifies that *exactly* '#' words must appear between the two surrounding dexpressions. `TEAM_NAME`, `SCORE`, `POS`, `ORD`, and `N_WIN` are all defined in `ExpFile` (Section 2.1.1).

EXAMPLE 3.

```
$cat sample | crep -e 'TEAM_NAME 2- TEAM_NAME 0= SCORE 2- POSS
0- ORD 2- N_WIN' -D ExpFile -w
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression:
{TEAM_NAME}([^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){TEAM_NAME}[^@]+{SCORE}([^@
]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){POSS}[^@]+{ORD}([^@]+|[^@]+@[^@]+|[^@]+@[
^@]+@[^@]+){N_WIN}
Compiling the flex file...
Executing Flex File...

Matching sentences:

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns
132-103 for their 15th consecutive home victory.


{TEAM_NAME}([^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){TEAM_NAME}[^@]+{SCORE}([^@
]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){POSS}[^@]+{ORD}([^@]+|[^@]+@[^@]+|[^@]+@[
^@]+@[^@]+){N_WIN}: the@AT Celtics@NNS defeated@VBD the@AT Phoenix@NPNP
Suns@NPNP 132-103@CD for@IN their@JJ 15th@CD consecutive@JJ home@NN
victory@NN
```

```
--------------------------------------------------------------------------

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the
Washington Bullets 118-94 for their third straight win.

{TEAM_NAME}([^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){TEAM_NAME}[^@]+{SCORE}([^@
]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){POSS}[^@]+{ORD}([^@]+|[^@]+@[^@]+|[^@]+@[
^@]+@[^@]+){N_WIN}: the@AT Chicago@NPNP Bulls@NPNP coasted@VBD past@IN
the@AT Washington@NPNP Bullets@NPNP 118-94@CD for@IN their@JJ third@CD
straight@RB win@VB

--------------------------------------------------------------------------

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past
the Washington Bullets 117-91 for its fifth straight victory.

{TEAM_NAME}([^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){TEAM_NAME}[^@]+{SCORE}([^@
]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){POSS}[^@]+{ORD}([^@]+|[^@]+@[^@]+|[^@]+@[
^@]+@[^@]+){N_WIN}: the@AT Suns@NPNP coasted@VBD past@IN the@AT
Washington@NPNP Bullets@NPNP 117-91@CD for@IN its@JJ fifth@CD straight@RB
victory@NN

--------------------------------------------------------------------------
```

In the interest of understanding what is going on here, note in the last matching sentence that there are two words in between 'The Suns' and 'the Washington Bullets' ('coasted' and 'past'), and that a maximum of *two* words were allowed by the expression ('`TEAM_NAME 2- TEAM_NAME`') due to the '`2-`' operator. The reader should now verify that the rest of the matching phrase indeed matches the given regular expression.

The expression in example 3 was translated from the **crep** syntax

```
'TEAM_NAME 2- TEAM_NAME 0= SCORE 2- POSS 0- ORD 2- N_WIN'
```

into the following `lex` syntax:

```
{TEAM_NAME}([^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){TEAM_NAME}[^@]+{SCORE}([^@
]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){POSS}[^@]+{ORD}([^@]+|[^@]+@[^@]+|[^@]+@[
^@]+@[^@]+){N_WIN}
```

One can clearly see the advantage of using **crep's** expression syntax for a search such as this one. One may, however, override **crep** syntax and use straight `lex` syntax to specify the expression; this option will be covered in example 13 in Section 3.

A HELPFUL TIP

The operator '0=', which means 'exactly zero words are allowed between' is logically equivalent to and produces the same result as '0-', which means 'at most zero words are allowed between'.

2.2.3.2 The ';' operator

The next example demonstrates the ';' operator. ';' is used when two expressions need to appear in the same sentence, but relative order in the sentence does not matter.

EXAMPLE 4.

```
cat sample | crep -e 'TEAM_NAME ; N_WIN' -D ExpFile -w
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {TEAM_NAME}
Regular Expression: {N_WIN}
Compiling the flex file...
Executing Flex File...

Matching sentences:

ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Charlotte took its third straight win in a 123-111 victory over the
Atlanta  Hawks.

   {N_WIN}: win@VB
   {N_WIN}: victory@NN
   {TEAM_NAME}: the@AT Atlanta@NPNP Hawks@NPNP

---------------------------------------------------------------------------

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the  Celtics defeated the  Phoenix  Suns
132-103 for their 15th consecutive home victory.

   {TEAM_NAME}: the@AT Celtics@NNS
   {TEAM_NAME}: the@AT Phoenix@NPNP Suns@NPNP
   {N_WIN}: victory@NN

---------------------------------------------------------------------------

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
```

```
points Wednesday night and the  Chicago  Bulls coasted past the
Washington  Bullets 118-94 for their third straight win.

   {TEAM_NAME}: the@AT Chicago@NPNP Bulls@NPNP
   {TEAM_NAME}: the@AT Washington@NPNP Bullets@NPNP
   {N_WIN}: win@VB

-------------------------------------------------------------------------

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the  Suns coasted past the
Washington  Bullets 117-91 for its fifth straight victory.

   {TEAM_NAME}: the@AT Suns@NP
   {TEAM_NAME}: the@AT Washington@NPNP Bullets@NPNP
   {N_WIN}: victory@NN

--------------------------------------------------------------------------

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the
Phoenix  Suns extended their winning streak to six games with a 123-109
victory over the  Minnesota  Timberwolves.

   {TEAM_NAME}: the@AT Phoenix@NPNP Suns@NPNP
   {N_WIN}: victory@NN
   {TEAM_NAME}: the@AT Minnesota@NPNP Timberwolves@NPNP

--------------------------------------------------------------------------
```

The two separate expressions (`TEAM_NAME` and `N_WIN`) match when they are found in any order in the sentence relative to one another.

2.2.3.3 '#+', '|', and parenthesis operators

The next example introduces the '|' , '#+' and parenthesis operators. The '|' operators is always used with parenthesis; together, they specify 'exclusive or' semantics. The '#+' operator specifies that *at least* '#' number of words must appear between the two surrounding expressions.

EXAMPLE 5.

```
$cat sample | crep -e 'TEAM_NAME 5+ (STRAIGHT|SCORE) 0- N_WIN' -
D ExpFile -w
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression:
{TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|{SCORE})[^@]+{N_WIN}
```

```
Compiling the flex file...
Executing Flex File...

Matching sentences:

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the  Chicago  Bulls coasted past the Washington
Bullets 118-94 for their third straight win.

   {TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|{SCORE})[^@]+{N_WIN}: the@AT
Chicago@NPNP Bulls@NPNP coasted@VBD past@IN the@AT Washington@NPNP
Bullets@NPNP 118-94@CD for@IN their@JJ third@CD straight@RB win@VB

---------------------------------------------------------------------------

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the  Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory.

   {TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|{SCORE})[^@]+{N_WIN}: the@AT
Suns@NPNP coasted@VBD past@IN the@AT Washington@NPNP Bullets@NPNP 117-91@CD
for@IN its@JJ fifth@CD straight@RB victory@NN

---------------------------------------------------------------------------

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the
Phoenix  Suns  extended their winning streak to six games with a 123-109
victory over the Minnesota Timberwolves.

{TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|{SCORE})[^@]+{N_WIN}: the@AT
Phoenix@ NPNP Suns@NPNP extended@VBD their@JJ winning@JJ streak@NN to@IN
six@CD games@NNS with@IN a@AT 123-109@CD victory@NN

---------------------------------------------------------------------------
```

Keep in mind that `lex` always matches the longest string possible. The sentence beginning with 'CHICAGO' matched, even though 'The Washington Bullets' appeared only four words in front of 'straight', one less than the required threshold by the operator '5+', because earlier in the sentence, another team name ('The Chicago Bulls') appeared, which is more than 5 words in front of 'straight'.

2.2.3.4 The '@@@' operator

The next example uses the '`@@@`' operator. '`@@@`' should be used if there is not a variable definition for the `lex` expression the user wishes to utilize, or if the user does not wish to utilize **crep** expression syntax for a portion of the expression. Anything appearing between '`@@@`'s gets dropped literally into the resulting `lex` rule without being touched by the **crep** expression translator.

Doing away with **crep** syntax altogether in favor of only `lex` syntax can be accomplished by enclosing the whole parameter of `-e` in '@@@'s. Section 2.3.1 offers another solution to this preference.

Even though `SCORE` appears in the variable definition file, we will demonstrate the '`@@@`' operator by substituting the `lex` expression '`([0-9]+-[0-9]+@)`' for `SCORE` in the expression of Example 5:

EXAMPLE 6.

```
$cat sample | crep -e 'TEAM_NAME 5+ (STRAIGHT|@@@([0-9]+-[0-
9]+@)@@@) 0- N_WIN' -D ExpFile -w
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|([0-9]+-[0-
9]+@))[^@]+{N_WIN}
Compiling the flex file...
Executing Flex File...

Matching sentences:

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win.

   {TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|([0-9]+-[0-9]+@))[^@]+{N_WIN}:
the@AT Chicago@NPNP Bulls@NPNP coasted@VBD past@IN the@AT Washington@NPNP
Bullets@NPNP 118-94@CD for@IN their@JJ third@CD straight@RB win@VB

---------------------------------------------------------------------------

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory.

   {TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|([0-9]+-[0-9]+@))[^@]+{N_WIN}:
the@AT Suns@NPNP coasted@VBD past@IN the@AT Washington@NPNP Bullets@NPNP
117-91@CD for@IN its@JJ fifth@CD straight@RB victory@NN

---------------------------------------------------------------------------

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the
Phoenix Suns extended their winning streak to six games with a 123-109
victory over the Minnesota Timberwolves.

   {TEAM_NAME}[^@]+(@[^@]+){5,}({STRAIGHT}|([0-9]+-[0-9]+@))[^@]+{N_WIN}:
the@AT Phoenix@NPNP Suns@NPNP extended@VBD their@JJ winning@JJ streak@NN
```

```
to@IN six@CD games@NNS with@IN a@AT 123-109@CD victory@NN

---------------------------------------------------------------------------
```

We should notice that the matching sentences of the last two examples is identical, even though we used a `lex` variable definition for SCORE in one example, and a literal `lex` expression in the other.

> ***A HELPFUL TIP***
>
> The '@@@' operator should always be used in pairs; otherwise, it makes no sense as an operator.

2.2.3.5 '@BEG@' and '@END@'

The next two examples demonstrate the `@BEG@` and `@END@` operators. Example 7 will use `@BEG@` to search for the string 'UPI' when it is the third word of a sentence.

EXAMPLE 7.

```
$cat sample | crep -e '@BEG@ 2= UPI' -w -D ExpFile
Parsing Your Regular Expression(s)...
crep: Expression Parser Warning:
  Expression `UPI` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: ^[^@]*@[^@]+@[^@]+UPI
Compiling the flex file...
Executing Flex File...

Matching sentences:

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory.

   ^[^@]*@[^@]+@[^@]+UPI:
BOSTON@NPNP (@( UPI

---------------------------------------------------------------------------

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win.
```

```
   ^[^@]*@[^@]+@[^@]+UPI:
CHICAGO@NPNP (@( UPI

----------------------------------------------------------------------------
```

LANDOVER (UPI) Jeff Hornacek and Cedric Ceballos scored 20 points each to lead six Suns in double figures Friday night as the Suns coasted past the Washington Bullets 117-91 for its fifth straight victory.

```
   ^[^@]*@[^@]+@[^@]+UPI:
LANDOVER@NPNP (@( UPI

----------------------------------------------------------------------------
```

MINNEAPOLIS (UPI) -- Tom Chambers scored 28 points Sunday and the Phoenix Suns extended their winning streak to six games with a 123-109 victory over the Minnesota Timberwolves.

```
   ^[^@]*@[^@]+@[^@]+UPI:
MINNEAPOLIS@NPNP (@( UPI

----------------------------------------------------------------------------
```

Note that the parenthesis count as words; this is a result of the behavior of `pos`. Example 8 introduces the use of '`@END@`'. We will query the corpus for all sentences which end with an `N_WIN`.

EXAMPLE 8.

$cat sample | crep -e 'N_WIN 1= @END@' -w -D ExpFile

```
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {N_WIN}[^@]*@[^@]+@@@$
Compiling the flex file...
Executing Flex File...

Matching sentences:
```

BOSTON (UPI) -- Kevin McHale 's 23 points led six Boston players in double figures Friday night as the Celtics defeated the Phoenix Suns 132-103 for their 15th consecutive home **victory.**

```
   {N_WIN}[^@]*@[^@]+@@@$: victory@NN .@. @@@

----------------------------------------------------------------------------
```

CHICAGO (UPI) -- Michael Jordan led a balanced scoring attack with 24 points Wednesday night and the Chicago Bulls coasted past the Washington Bullets 118-94 for their third straight **win.**

```
{N_WIN}[^@]*@[^@]+@@@$: win@VB .@. @@@
```

```
LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory.

   {N_WIN}[^@]*@[^@]+@@@$: victory@NN .@. @@@
```

Why did we use the expression ‘`N_WIN 1= @END@`’ to search for the last word of a sentence, and not ‘`N_WIN 0= @END@`’? Because just as parenthesis count as words, so does the period which ends the sentence (note that the tagger gave the period a period tag). We must allow for it.

Also note that the matches ended with ‘@@@’. It is simply **crep’s** way of telling when an end of sentence exists. The ‘@@@’ should not be confused with the escape operators (‘@@@’; Section 2.2).

Note: only one '@BEG@' and one '@END@' operator are allowed in the same expression. The following expression is illegal:

```
(TEAM_NAME 0- SCORE 2= @END@ | ORD 0- STRAIGHT 3= @END@)
```

It should be rewritten as follows:

```
(TEAM_NAME 0- SCORE 2=| ORD 0- STRAIGHT 3=) @END@
```

2.2.3.6 The ‘.’ operator

In the last example in this subsection, we will demonstrate the ‘.’ operator. The ‘.’ operator is used if we don’t care how many words appear in between the two adjacent expressions, so long as they appear in that order.

EXAMPLE 9.

```
$cat sample | crep -e 'TEAM_NAME . SCORE . TEAM_NAME' -D ExpFile
-w
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {TEAM_NAME}.*{SCORE}.*{TEAM_NAME}
```

```
Compiling the flex file...
Executing Flex File...

Matching sentences:

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the
Phoenix  Suns extended their winning streak to six games with a 123-109
victory over the  Minnesota  Timberwolves.

   {TEAM_NAME}.*{SCORE}.*{TEAM_NAME}: the@AT Phoenix@NPNP Suns@NPNP
extended@VBD their@JJ winning@JJ streak@NN to@IN six@CD games@NNS with@IN
a@AT 123-109@CD victory@NN over@IN the@AT Minnesota@NPNP Timberwolves@NPNP

------------------------------------------------------------------------------
```

Notice that the sentence starting with 'ATLANTA' in our corpus sample did not match because **crep** did not recognize 'Charlotte' as a TEAM_NAME due to TEAM_NAME's definition in ExpFile (Section 2.1.1).

2.2.3.7 The '?' operator

The **crep** operator '?' works analagously to its counterpart in lex; the expression operated on is optional.

EXAMPLE 10.

```
$cat  sample  |  crep  -e  'STRAIGHT  0-  (home@  0-)?  N_WIN'  -w  -D
ExpFile
Parsing Your Regular Expression(s)...
crep: Expression Parser Warning:
  Expression `home` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {STRAIGHT}[^@]+(home@[^@]+)?{N_WIN}
Compiling the flex file...
Executing Flex File...

Matching sentences:


ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Carlotte took its third straight win in a 123-111 victory over the
Atlanta Hawks.

   {STRAIGHT}[^@]+(home@[^@]+)?{N_WIN}: straight@RB win@VB
```

```
----------------------------------------------------------------------------

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory.

   {STRAIGHT}[^@]+(home@[^@]+)?{N_WIN}: consecutive@JJ home@NN victory@NN

----------------------------------------------------------------------------

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win.

   {STRAIGHT}[^@]+(home@[^@]+)?{N_WIN}: straight@RB win@VB

----------------------------------------------------------------------------

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory.

   {STRAIGHT}[^@]+(home@[^@]+)?{N_WIN}: straight@RB victory@NN

----------------------------------------------------------------------------
```

2.3 Using a variable definition file in crep expression syntax: the -d option

Now that the **crep** expression syntax has been introduced, we may utilize its simplicity in the variable definition file. Following is a variable definition file in **crep** expression syntax (`ExpFile2`). It contains all the definitions of `ExpFile` presented in Section 2.1.1, and a few new definitions. Essentially, `ExpFile2` carries all the regular-expression matching power as the `lex`-syntax `ExpFile` plus the power of two new definitions: `COMBO` and `SUPERCOMBO`.

EXAMPLE 11.

```
$cat  ExpFile2
SPACE               @@@[ ]+@@@

SUPERCOMBO          COMBO 4- STRAIGHT

COMBO               TEAM_NAME 0- SCORE

TEAM_NAME           @@@([tT]he@AT[ ][A-Z][a-z][a-zA-Z]+@NNS)|([tT]he@AT[ ][A-
Z][a-z][a-zA-Z]+@NP)|([tT]he@AT[ ][A-Z][a-z][a-zA-Z]+@NPNP[ ][A-Z][a-z][a-
zA-Z]+@NPNP)@@@

SCORE               @@@[0-9]+@CD{SPACE}[0-9]+@CD|[0-9]+@CD{SPACE}(to@
```

```
[A-Z]+){SPACE}[0-9]+@CD|[0-9]+-[0-9]+@CD@@@

N_WIN              ((victory|win|triumph|decision|romp)@NN)|(win@VB)

LOSS               (loss|defeat|upset|setback)@NN

STRAIGHT           (straight|consecutive)@

POSS               (my|your|her|his|its|our|their)@JJ

VBD_WIN            @@@((defeated|beat|downed|edged|pounded|routed|minced|
troubled|dumped|crushed|stopped|outlasted|upended|blasted)@)|(blew@[A-Z]+[
]out@)|(coasted@VB[A-Z][ ]past@)|(held@[A-Z]+[ ]off@)|(knocked@[A-Z]+
off@)@@@

ORD               @@@((first|second|third|fourth|fifth|sixth|seventh|eighth|
ninth|tenth)@CD)|([1-9][0-9]*((0|[4-9])th)|1st|2nd|3rd)@@@@

VBG_ENABLE        (helping|pacing|guiding|lifting|sparking|igniting|topping|
powering|rallying|propelling|sending|carrying|engineering|spurring|pushing|
fueling|bringing)@VB
```

Notice that some definitions, such as `POSS` and `STRAIGHT`, are identical to their counterparts in `ExpFile`. Others, such as `ORD` and `SCORE`, contain the `lex` definitions of `ExpFile` between the '@@@' operators[6]. Finally, we see two definitions which utilize other **crep** operators: `COMBO` and `SUPERCOMBO`, which give the user more macro power. With `ExpFile2` specified to **crep** using the `-d` option, the user may use '`SUPERCOMBO`' in the `-e` parameter rather than its expanded substitution value, which is '`TEAM_NAME 0- SCORE 4- STRAIGHT`' (The previous expression is, of course, still in **crep** syntax, and is translated into `lex` syntax at execution time, for another level of abstraction).

Note: every **crep** expression operator may be used in a variable definition file *except* the semicolon (';') operator.

EXAMPLE 12.

```
$cat sample | crep -e 'SUPERCOMBO' -d ExpFile2 -w
Parsing Your Regular Expression(s)...
Parsing your Variable Definition File...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {SUPERCOMBO}
Compiling the flex file...
Executing Flex File...
```

[6] Why do some definitions require the '@@@'s and some do not? Since **crep** recognizes characters such as '-' and '+' to hold special meaning, expressions containing them must be surrounded by the '@@@' operator if those characters are not being used as operators.

Matching sentences:

BOSTON (UPI) -- Kevin McHale 's 23 points led six Boston players in double figures Friday night as the Celtics defeated **the Phoenix Suns 132-103 for their 15th consecutive** home victory.

{SUPERCOMBO}: the@AT Phoenix@NPNP Suns@NPNP 132-103@CD for@IN their@JJ 15th@CD consecutive@

CHICAGO (UPI) -- Michael Jordan led a balanced scoring attack with 24 points Wednesday night and the Chicago Bulls coasted past **the Washington Bullets 118-94 for their third straight** win.

{SUPERCOMBO}: the@AT Washington@NPNP Bullets@NPNP 118-94@CD for@IN their@JJ third@CD straight@

LANDOVER (UPI) Jeff Hornacek and Cedric Ceballos scored 20 points each to lead six Suns in double figures Friday night as the Suns coasted past **the Washington Bullets 117-91 for its fifth straight** victory.

{SUPERCOMBO}: the@AT Washington@NPNP Bullets@NPNP 117-91@CD for@IN its@JJ fifth@CD straight@

---

3. MORE OPTIONS OF crep

This section will introduce other available command line options of **crep**.

3.1 Other options for expression passing

3.1.1 -*E:* straight `lex` syntax instead of the crep expression syntax

If one wishes to use `lex` syntax exclusively, and would prefer not to escape `lex` expressions with the '`@@@`' operators, the `-E` option may be used in place of the `-e` option. `-E` passes the *entire* expression to `lex` -- no parsing is done by **crep**.

EXAMPLE 13.

```
$cat sample | crep -E '{TEAM_NAME}.*{SCORE}.*{TEAM_NAME}' -D
ExpFile -w
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {TEAM_NAME}.*{SCORE}.*{TEAM_NAME}
Compiling the flex file...
```

```
Executing Flex File...

Matching sentences:

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the
Phoenix  Suns extended their winning streak to six games with a 123-109
victory over the  Minnesota  Timberwolves.

   {TEAM_NAME}.*{SCORE}.*{TEAM_NAME}: the@AT Phoenix@NPNP Suns@NPNP
extended@VBD their@JJ winning@JJ streak@NN to@IN six@CD games@NNS with@IN
a@AT 123-109@CD victory@NN over@IN the@AT Minnesota@NPNP Timberwolves@NPNP

---------------------------------------------------------------------------
```

We should notice that the matching output is identical to that of Example 9. The only difference in execution of **crep** is that the expression parsing module was skipped (i.e. there was no line '`Parsing Your Regular Expression(s)...`' printed).

The user may specify ANY `lex` regular expression with the `-E` option EXCEPT the space character. This limitation can be circumvented by the following method. Enter the following definition into a variable definition file:

```
SPACE                [ ]
```

and then change an arbitrary expression on the command line which uses the above variable definition file as follows:

```
from       -E '[0-9]+@CD[ ][0-9]+@CD'
                        ^^^
                        Note the space character

to         -E '[0-9]+@CD{SPACE}[0-9]+@CD'
                        ^^^^^^^
                        Using the SPACE definition.
```

3.1.2 *-f* and *-F*

If one wishes to edit the expression in a file instead of on the command line (useful if the expression grows lengthy and complex), one may do so with `-f` and `-F` (`-f` for specifying **crep** syntax, and `-F` for `lex` syntax):

EXAMPLE 14.

```
$cat  myfile
TEAM_NAME 5+ (STRAIGHT|@@@([0-9]+-[0-9]+@)@@@) 0- N_WIN
$cat  sample  |  crep  -f  myfile  -D  ExpFile  -w
```

This command will produce output identical to that of example 5. Similarly, the user may read in a `lex` expression from a file with the `-F` option:

EXAMPLE 15.

```
$cat  myfile2
{TEAM_NAME}.*{SCORE}.*{TEAM_NAME}
$cat  sample  |  crep  -F  myfile  -D  ExpFile  -w
```

We should note that the matching sentences of the above command are the same as that of Example 9. Note that the use of single quotes around the expression when it appears in a file is not necessary as it is on the command line.

3.1.3 *-m*

The `-m` option is used for batch-style expression matching, and, as the reader will see, for special searches such as negation in Section 3.1.3.1. With `-m`, the user can specify *n* expressions, *n* outfile names, and have each expression E_i redirect its matches from the corpus into the file F_i:

EXAMPLE 16.

```
$cat  sample   |  crep  -m  'Celtics  file1  VBG_ENABLE  file2
@@@{STRAIGHT}.*{N_WIN}@@@  file3'  -D  ExpFile
Tagging words in your corpus file(s)...
Building the flex file...
crep: Expression Parser Warning:
  Expression `Celtics` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Regular Expression 1: Celtics
Regular Expression 2: {VBG_ENABLE}
Regular Expression 3: {STRAIGHT}.*{N_WIN}
Compiling the flex file...
Executing Flex File...

The output is being sent to the appropriate file(s).
$cat  file1
BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
```

```
103 for their 15th consecutive home victory.

$cat  file2
$cat  file3
ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Charlotte took its third straight win in a 123-111 victory over the
Atlanta Hawks.

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory.

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win.

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory.
$
```

There are two factors to keep in mind when using `-m`:

1) All expressions used with the `-m` option are run through the **crep** expression parser (even though no parser message appears), which should not be a problem for using straight `lex` expressions: simply escape the whole expression with the ‘`@@@`’ operators for `lex`-exclusive searches.

2) Spaces cannot be used on the command line in the middle of an expression with -m. Therefore, to execute an ambiguous command like

```
$cat sample | crep -m 'STRAIGHT 0= N_WIN file1 SCORE 1- POSS
file2' -D ExpFile
```

substitute a ‘-’ wherever a space should appear in the expression, as follows:

```
$cat sample | crep -m 'STRAIGHT-0=-N_WIN file1 SCORE-1--POSS
file2' -D ExpFile
```

3.1.3.1 Special searching capabilities of -m

The `-m` option makes possible two kinds of special searches: negation and at most/at least/exactly semantics for whole expressions. The techniques are described below.

3.1.3.1.1 Negation searches

Negation can be achieved by simultaneously searching for the two following expressions: one that matches every sentence, and one that matches exactly what is to be discarded. After the two respective match files have been created, use of the `diff` tool will return what is desired.

Say, for example, the user wishes to query the toy corpus `sample` for all the sentences which do not contain `STRAIGHT`:

EXAMPLE 17.

```
$cat sample | crep -m '@@@.@@@ everything STRAIGHT junk' -D
ExpFile
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression 1: .
Regular Expression 2: {STRAIGHT}
Compiling the flex file...
Executing Flex File...

The output is being sent to the appropriate file(s).
$cat everything
ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Charlotte took its third straight win in a 123-111 victory over the
Atlanta Hawks .

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory .

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win .

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory .

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the Phoenix
Suns extended their winning streak to six games with a 123-109 victory over
the Minnesota Timberwolves .

$cat junk
ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Charlotte took its third straight win in a 123-111 victory over the
Atlanta Hawks .

BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
```

```
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory .

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win .

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory .
```

*Only the sentence starting with ''*Minneapolis' *does not appear in* `junk` -- *that is the one sentence we want. After* `everything` *and* `junk` *have been created, we* `diff` *them and pass* `diff`*'s output through the* **crep** *tool* **diff_clean** *to get what we want without the '*<*'s output by* `diff`*:*

```
$diff everything junk | diff_clean
 MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the
Phoenix Suns extended their winning streak to six games with a 123-109
victory over the Minnesota Timberwolves .
$
```

Warning: Use of `-w` with negation searches may be dangerous, since the expression '`@@@.@@@`' matches *every character* that is not part of the expression the user does not want. Furthermore, `-w` does not make any sense with a negation search of this type, because the semantics of the search are "return all *sentences* which do not contain expression *x*." In other terms, using `-w` with `-m` negation is analagous to querying a database for every field which does not match 'zucchini'; unless you are querying some specialized vegetarian corpus, the size of the found set will be a function of the size of the corpus itself.

Negations in which one wishes to search for all sentences containing 'foo' *not* followed by 'bar'can also be realized using **crep** in the following way: search for two two expressions -- the common part as one expression (foo), and 'foo' followed by 'bar' as the other epxression. Here we utilize the `lex` property that imbedded matches result in the longest expression matching only.

In example 18, we want all sentences which contain a `STRAIGHT` (recall that `STRAIGHT` stands for either 'straight' or 'consecutive') which is not directly followed by the word 'win':

<u>EXAMPLE 18.</u>

```
$cat sample | crep -m 'STRAIGHT wanted STRAIGHT-0--win@ junk' -D
ExpFile
```

```
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression 1: {STRAIGHT}
crep: Expression Parser Warning:
  Expression `win` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Regular Expression 2: {STRAIGHT}[^@]+win@
Compiling the flex file...
Executing Flex File...

The output is being sent to the appropriate file(s).
$cat  wanted
BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive  home victory .

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight  victory .

$cat  junk
ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Charlotte took its third straight  win in a 123-111 victory over the
Atlanta Hawks .

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight  win .
```

Note that `junk` does not represent *all* sentences that we don't want; sentences that did not contain at least a `STRAIGHT` were omitted from both matching sets.

3.1.3.1.2 'At most', 'at least', and 'exactly' semantics

The default semantics of an expression used in **crep** is 'at least one'. For instance, in example 4, we searched for `N_WIN`, and all sentences with at least one `N_WIN` were returned. But what if we would like to extract only the sentences with *at most* one `N_WIN`? The `-m` option realizes this need.

To do so, we will give two expressions to `-m`: the overqualified expression (`N_WIN-.-N_WIN`) which returns all sentences containing at least two `N_WIN`'s; and the expression we want: simply `N_WIN`. We once again utilize the `lex` property that imbedded matches result in the longest expression matching only. Without the first expression (`N_WIN-.-N_WIN`) to 'steal away' what we don't want, `N_WIN` would take all of the sentences containing at least one `N_WIN`.

EXAMPLE 19.

```
$cat sample | crep -m 'N_WIN-.-N_WIN junk N_WIN wanted' -D
ExpFile -w
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression 1: {N_WIN}.*{N_WIN}
Regular Expression 2: {N_WIN}
Compiling the flex file...
Executing Flex File...

The output is being sent to the appropriate file(s).
$cat junk
ATLANTA ( UPI ) -- Dell Curry scored 20 points and Kelly Tripucka added 18
as Charlotte took its third straight win in a 123-111 victory over the
Atlanta Hawks .

   {N_WIN}.*{N_WIN}: win@VB in@IN a@AT 123-111@CD victory@NN

---------------------------------------------------------------------------

$cat wanted
BOSTON ( UPI ) -- Kevin McHale 's 23 points led six Boston players in
double figures Friday night as the Celtics defeated the Phoenix Suns 132-
103 for their 15th consecutive home victory .

   {N_WIN}: victory@NN

---------------------------------------------------------------------------

CHICAGO ( UPI ) -- Michael Jordan led a balanced scoring attack with 24
points Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win .

   {N_WIN}: win@VB

---------------------------------------------------------------------------

LANDOVER ( UPI ) Jeff Hornacek and Cedric Ceballos scored 20 points each to
lead six Suns in double figures Friday night as the Suns coasted past the
Washington Bullets 117-91 for its fifth straight victory .

   {N_WIN}: victory@NN

---------------------------------------------------------------------------

MINNEAPOLIS ( UPI ) -- Tom Chambers scored 28 points Sunday and the Phoenix
Suns extended their winning streak to six games with a 123-109 victory
over the Minnesota Timberwolves .

   {N_WIN}: victory@NN

---------------------------------------------------------------------------
```

```
$
```

This technique works for longer expressions as well (say, for finding all sentences with *exactly* two `N_WIN`'s). But in order for this to work, the longer expression MUST appear before the shorter expression on the command line. This is due to the fact that `lex` will match the same string for '`foo-.-foo`' as it does for '`foo-.-foo-.-foo`'. If '`foo-.-foo-.-foo`' appears *before* '`foo-.-foo`' on the command line, the search will work, and '`foo-.-foo`' will get the sentences containing *exactly* two `foo`'s and `foo-.-foo-.-foo` will get all sentences containing three or more `foo`'s. Sentences with only one `foo` are not found by either expression.

To search for *at most* two `foo`'s, replace `foo-.-foo` in the above search by just plain `foo`. Here `foo` will match any sentence containing at least one `foo`, unless the sentence is stolen away by `foo-.-foo-.-foo`. In other words, `foo`, when searched for at the same time as `foo-.-foo-.-foo`, will return all sentences with at most two `foo`'s.

In any case, use of `-w` when searching with at most/at least/exactly semantics will provide valuable insight as to what is matching and what is not.

3.1.4 *-g:* Reading the *-m* parameter from a file

Analagous to `-f` and `-F`, `-g` allows the user to edit the parameter of `-m` in a file. `-g` will work properly so long as each E_i is separated by at least one whitespace character (space, tab, or newline) from each F_i, and vice versa.

3.2 Options for increasing the execution speed of crep

3.2.1 *-k* and *-x*

The compile time for expressions is a function, among other factors, of the expression complexity. If the user is searching different corpora with the same expression AND the same variable definition file repeatedly, `-k` and `-x` may be used. When `-k` is used, **crep** is instructed to keep all temporary files rather than perform the default action, which is to destroy them all. If **crep** is called *directly after* a call to **crep** was made with the `-k` option, `-x` (the express option) may be used to utilize the already compiled temporary files; therefore, no compilation will take place. We shall use `-k`, and then both `-k` and `-x`, on the command from Example 3, running `time` on both trials:

EXAMPLE 20.

```
$ time cat sample | crep -e 'TEAM_NAME 2- TEAM_NAME 0= SCORE 2-
POSS 0- ORD 2- N_WIN' -D ExpFile -w -k
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression:
{TEAM_NAME}([^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){TEAM_NAME}[^@]+{SCORE}([
^@]+|[^@]+@[^@]+|[^@]+@[^@]+@[^@]+){POSS}[^@]+{ORD}([^@]+|[^@]+@[^@]+|[^@]+
@[^@]+@[^@]+){N_WIN}
Compiling the flex file...
Executing Flex File...
```

{ matching sentence output is same as from Example 3 }

```
The temporary files have been saved in your current directory.
You may use the -x option the next time you run crep to
use the precompiled flex file, avoiding compilation time.

To destroy all temp files, type crep_clean.

real    0m30.36s
user    0m20.36s
sys     0m9.45s
$ time cat sample | crep -e 'TEAM_NAME 2- TEAM_NAME 0= SCORE 2-
POSS 0- ORD 2- N_WIN' -D ExpFile -w -k -x
Tagging words in your corpus file(s)...
Executing Flex File...
```

{ output is same as from Example 3 }

```
The temporary files have been saved in your current directory.
You may use the -x option the next time you run crep to
use the precompiled flex file, avoiding compilation time.

To destroy all temp files, type crep_clean.

real    0m5.70s
user    0m0.56s
sys     0m4.85s
$
```

Note that in the second call to **crep** (with the `-x` option) there was no expression parsing or `lex` source compiling; the executable from the previous call to **crep** was used.

crep_clean, a tool of **crep**, can be invoked to destroy all temporary files created by **crep** without having to run **crep** again. See Section 7 for more information about **crep_clean**.

3.2.2 *-n* and *-p:* tagged input files

Tagging large corpora may take a considerable amount of time. **crep** supports three options(`-n`, `-p`, and `-P`) and one tool (**crep_prep**; discussed in Section 7.2) to give the user an ability to tag a corpus only once and save the tagged input as a file. The `-n <filename>` option will capture the tagged input and save it to a file which won't be destroyed by **crep** when the rest of the temporary files are deleted by default. To use the file created previously with the `-n` option, use the `-p` (pretagged) option as follows:

EXAMPLE 21.

First, we shall create the tagged file out of an untagged basketball lead-in corpus `big_one` (150K) with the `-n` option:

```
$time cat big_one | crep -e 'TEAM_NAME 0- SCORE' -D ExpFile -n
big.tagged
Parsing Your Regular Expression(s)...
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: {TEAM_NAME}[^@]+{SCORE}
Compiling the flex file...
Executing Flex File...

Matching sentences:
```

{6 pages of output skipped}

```
Tagged input was saved in a file under the name you specified with
the -n parameter.

real    0m59.95s
user    0m26.48s
sys     0m24.35s
```

Now, we can call **crep** *with our newly created* `big.tagged` *file. The* `-p` *option tells* **crep** *to skip the tagging module (the modules are explained in Section 4). We may freely change the expression, but in the following example, we will use the same expression for an accurate time comparision.*

```
$time cat big.tagged | crep -p -e 'TEAM_NAME 0- SCORE' -D
ExpFile
Parsing Your Regular Expression(s)...
Building the flex file...
Regular Expression: {TEAM_NAME}[^@]+{SCORE}
Compiling the flex file...
Executing Flex File...
```

```
Matching sentences:
```

{6 pages of output skipped}

```
real    0m28.31s
user    0m15.70s
sys     0m11.23s
$
```

3.3 Other options

3.3.1 *-c* and *-P*

Alternatively, input to **crep** can be specified as a parameter of an option rather than from standard input. To input an untagged corpus, use `-c`. To input a tagged corpus, use `-P`.

EXAMPLE 22a.

For untagged corpora,

```
$cat big_one | crep -e 'TEAM_NAME 0- SCORE' -D ExpFile
```

yields the same results as

```
$crep -c big_one -e 'TEAM_NAME 0- SCORE' -D ExpFile
```

EXAMPLE 22b.

For tagged corpora,

```
$cat big.tagged | crep -p -e 'TEAM_NAME 0- SCORE' -D ExpFile
```

yields the same results as

```
$crep -P big.tagged -e 'TEAM_NAME 0- SCORE' -D ExpFile
```

3.3.2 *-t:* printing tagged output

Sometimes it is helpful to view the matching sentences with all tags intact. By default, **crep** strips them off and displays regular text. To output the sentences with tagged words, use `-t`:

EXAMPLE 23.

```
$cat sample | crep -e 'Celtics' -t
Parsing Your Regular Expression(s)...
crep: Expression Parser Warning:
  Expression `Celtics` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: Celtics
Compiling the flex file...
Executing Flex File...

Matching sentences:

BOSTON@NPNP (@( UPI@NPNP )@) --@: Kevin@NPNP McHale@NPNP 's@$ 23@CD
points@NNS led@VBD six@CD Boston@NP players@NNS in@IN double@JJ figures@NNS
Friday@NP night@NN as@CS the@AT Celtics@NNS defeated@VBD the@AT
Phoenix@NPNP Suns@NPNP 132-103@CD for@IN their@JJ 15th@CD consecutive@JJ
home@NN victory@NN .@.

$
```

4. TOOLS USED/CHAIN OF EXECUTION

This section discusses the interaction between the various tools and modules which make up **crep**. While knowledge of the various parts which comprise **crep** may not be directly necessary for the use of **crep**, the flow of execution may aid in understanding some of the underlying assumptions of **crep**.

4.1 The five modules

There are five main modules of **crep**: the expression parsing module, the tagging module, the `lex` compilation module, the `lex` execution module, and the output module. Their interaction with one another is illustrated in fig. 4-1. A description of each module follows:

The **Expression Parsing Module** parses an expression from **crep** expression syntax (the syntax is fully described in section 2) into `lex` syntax by means of `parser_exp`. If the user specified a `lex` variable definition file, `Name_Extractor` will fetch the variable names and `parser_exp` will substitute the appropriate `lex` variable(s) in the expression.

The Expression Parsing Module also passes the variable definition file specified by `-d` into the parser. Variable definition files specified by `-D` (`lex` syntax files) do not enter the Expression Parsing Module.

The **Tagging Module** performs three operations on the corpus input (unless the input is pre-tagged, in which case this module is skipped):

1) delineates the corpus into sentences (`sent` or a user-authored delimiter)
2) tags each part of speech with `pos`
3) transforms `pos` output into a format which **crep** can interpret (`pos_to_crep` and `eat`)

After tagging(2) and transforming(3), the character '`@`' separates words in the corpus from their associated `pos` tag, while the space character is used to separate word@tag pairs. The `pos` tag names are those produced by `pos` (Appendix D).

The expression from the Expression Parsing Module, along with the variable definition file specified by the user (if any) is fed into the **Lex Compilation Module**. Here the source `lex` file is built by `crepdriver`. The variable definition file forms the definitions section, and the regular expression(s) forms the rule section. The action(s) of the rule(s) are added here automatically.

The compiled `lex` file, `main_crep`, is then executed by the **Lex Execution Module**, its input being previously tagged, whether it was tagged by the **Tagging Module** or pre-tagged.

The output of the **Lex Execution Module** is then interpreted by the **Output Module**. The style of output depends on what options were fed into **crep** on the command line.

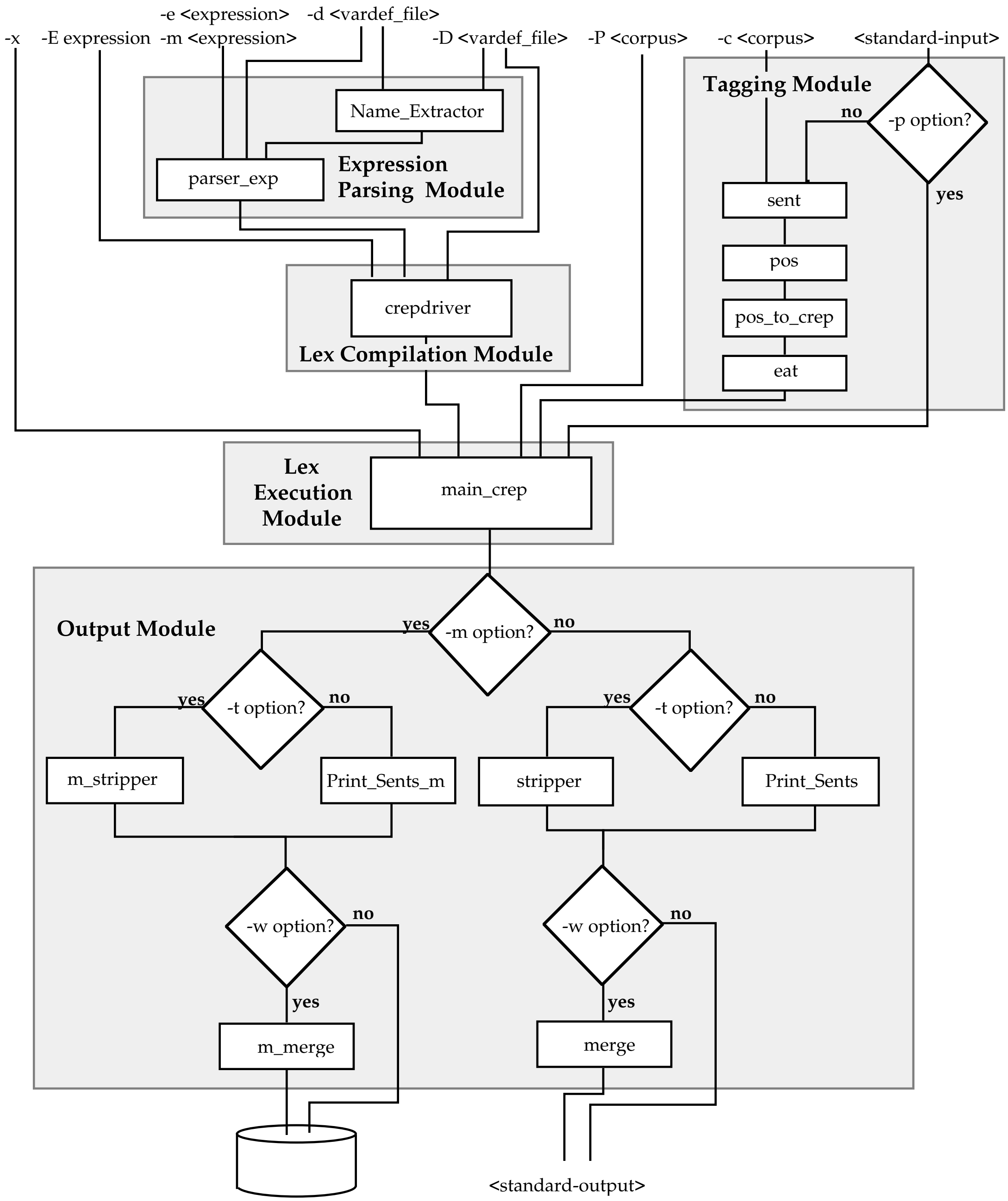

Figure 4-1. **crep** tools, modules, and chain of execution

5. TECHNIQUES FOR EFFICIENT USE OF crep

5.1 Getting familiar with what a tagged corpus looks like

Since most uses of **crep** require knowledge of part of speech tags, the user should become familiar with what a tagged corpus file looks like. That is to say, the user should be aware of how the input appears to **crep** after tagging it and converting it to **crep**-palatable syntax. Before the user dives in to **crep**ping, the best piece of advice to the novice user is to tag a corpus using **crep_prep** (Section 7.2) and take a look at it. Notice what **crep** does with certain words and also with punctuation. Often users fail to get **crep** working properly because they omit the tags in their expression, because there are no tags in the free text, and the obvious thing to search for is what they see in the raw text. Section 5.2 presents an example of what happens when a tag is omitted in one kind of search.

5.2 Stacking expressions to one's advantage

In order to properly form the expressions, the user must be aware that the actual matching of words in the corpus to the expression is done when the words in the corpus are *tagged*. Therefore, when one is forming an expression, whether it be on the command line or in a definition file, a tag should be appended to each word of the expression. For example, one may define 'color' in a variable definition file as follows:

```
COLOR          (red|blue|yellow|green|purple|teal|mauve)@JJ
```

This is an example of a properly-formed **crep** variable definition in `lex` syntax. Every 'color' on the right hand side is followed by a '@JJ', which is an example of a **crep** tag[7] . If one searched for 'purple' and not 'purple@JJ', the **crep** parser operators will not work properly. Since the operators search for '@'s to count the number of words, the tag of 'purple' will be counted as a separate word, and the **crep** expression 'purple 0- pickle' would not match the actual phrase 'purple pickle' in the corpus, since the tagged version of the phrase is 'purple@TAG1 pickle@TAG2', where TAG1 and TAG2 are chosen by `pos` depending on what part of speech `pos` thinks they are. The string '@TAG1', when encountered by **crep**, is counted as a separate word, and since the '0-' operator disallows any words in between the two expressions (Section 2.2), 'purple 0- pickle' does not match 'purple pickle' in the corpus. If the expression 'purple@JJ 0- pickle' is searched for, then the tagged corpus fragment 'purple@JJ pickle@NN' would match.

5.2.1 Ignoring parts of speech

If, however, the tagger tags the above corpus fragment as 'purple@NN pickle@NN', then the **crep** expression 'purple@JJ 0- pickle@NN' would not match. If one does not care about the part of speech of 'purple', and only insists that it is followed immediately by the word 'pickle' in the corpus, then the corresponding **crep** expression should be 'purple@ 0- pickle'. The tag is incomplete, but since the '@' is there, the rest of the tag will just be ignored and not treated as a whole word. Therefore, according to **crep**, the corpus words 'purple' and 'pickle' will match the expression no matter how they are tagged as long as they are adjacent and in that order.

5.2.2 Compensating for the tagger's errors

By examining the tagged version of a corpus, errors by the tagger can be circumvented by **crep** by making modifications to the variable definition file and/or expression.

In the toy corpus `sample` used in Section 2, the second sentence receives the following part-of-speech tags from `pos`:

```
CHICAGO@NPNP (@( UPI@NPNP )@) --@: Michael@NPNP Jordan@NPNP led@VBD a@AT
```

[7] **crep** tags are derived from `pos` tags. The abbreviations of the tags used by **crep** are the same as those assigned by `pos`; only the character separating the word and tag are different ('@' for **crep** and '/' for `pos`). For a complete list of which abbreviation stands for which part of speech, consult Appendix D.

```
balanced@VBN scoring@NN attack@NN with@IN 24@CD points@NNS Wednesday@NP
night@NN and@CC the@AT Chicago@NPNP Bulls@NPNP coasted@VBD past@IN the@AT
Washington@NPNP Nulls@NPNP 118-94@CD for@IN their@JJ third@CD straight@RB
win@VB.
```

Note that `pos` incorrectly tagged the word 'win' with 'VB', which is a `pos` tag for a verb. In section 2, we searched for the word 'win' as a noun in the variable definition

```
N_WIN            ((victory|win|triumph|decision|romp)@NN)|(win@VB)
```

and `N_WIN` matched this particular 'win' in examples 3, 4, and 5 of section 2 because we added the extra '(win@VB)' case on the end of our definition. Thus by adding extra cases to our definitions, we can compensate for tagger errors. This practice, however, may result in producing spurious output; the user must decide how to treat errors of this nature and choose whether the output should filter out some matching sentences, or if **crep** should let some incorrect sentences pass through to the output. In either case, judicious use of the `-w` and `-t` options can help the user decide what should be matching and what shouldn't.

5.3 Searching tricks

5.3.1 Part-of-speech searching

Since **crep** scans tagged text, the user can run very general discovery tests without having a variable definition file by simply specifying the part-of-speech tag in the expression rather than a specific word or variable definition. A variable definition file can then be built from the discovered words. For instance, to search for an adjective adjacent to a noun, simply search for the expression

```
@JJ 0- @NN
```

This expression would return all sentences which contain an adjective adjacent to and before a noun.

5.3.2 "Trapping"

A common discovery technique of using **crep** is called trapping. Trapping is the act of searching for a family of words by specifying common surrounding words and 'trapping' the desired unknown word or words in the middle. Say, for instance, we wished to know what kinds of 'defeating' verbs are used when an article reports that one basketball team defeats another. We can search one possible sentence structure, "trapping" the verbs which appear in between two team names:

EXAMPLE 24.

```
$cat big_one.tagged | crep -p -e 'TEAM_NAME 1- @@@[a-zA-Z]@VB
@@@ 1- TEAM_NAME' -w -D ExpFile
Parsing Your Regular Expression(s)...
Building the flex file...
Regular Expression:
{TEAM_NAME}([^@]+|[^@]+@[^@]+)[a-zA-Z]@VB([^@]+|[^@]+@[^@]+){TEAM_NAME}

Compiling the flex file...
Executing Flex File...

Matching sentences:

LOS ANGELES ( UPI ) -- Danny Manning scored 27 points and Ron Harper added
22 Thursday night , helping the Los Angeles Clippers down the Knicks
101-91 , snapping a 12-game losing streak at the hands of New York dating
back to February 23, 1986.

   {TEAM_NAME}([^@]+|[^@]+@[^@]+)[a-zA-Z]@VB([^@]+|[^@]+@[^@]+){TEAM_NAME}:
the@AT Los@NPNP Angeles@NPNP Clippers@NPNP down@VB the@AT Knicks@NNS

---------------------------------------------------------------------------

CHICAGO UPI Michael Jordan led a balanced scoring attack with 24 points
Wednesday night and the Chicago Bulls coasted past the Washington
Bullets 118-94 for their third straight win.

   {TEAM_NAME}([^@]+|[^@]+@[^@]+)[a-zA-Z]@VB([^@]+|[^@]+@[^@]+){TEAM_NAME}:
the@AT Chicago@NPNP Bulls@NPNP coasted@VBD past@IN the@AT Washington@NPNP
Bullets@NPNP

---------------------------------------------------------------------------

...
```

Note that since we only searched for the verb tag, the corresponding verbs themselves were captured in our neatly set trap. The verbs 'down' and 'coasted past' were "trapped" in between two `TEAM_NAME`'s, with a one word allowance between each `TEAM_NAME` and the verb. Since 'coasted past' is two words, it would not have matched if the allowance was '0-' instead of '1-' on both sides. By changing the word allowance operators (here we used '1-'), **crep** can yield different matches. The user can then add these verbs to the variable definition file.

6. CUSTOM SENTENCE DELIMITER AUTHORING TUTORIAL

One of the main features of **crep** is that it returns the entire sentence containing the matching expression, not just the physical line in the file or just the expression itself. By examining figure 4-1, the reader will notice that **crep** delimits the corpus into sentences with a sentence delimiter before anything else is done to the corpus. By default, this delimiting is done by the default delimiter tool which is part of **crep** (called `sent`). This delimiter, while containing several rules to determine where the ends of sentences should fall, does not cover all possible cases, especially for corpora in a specific domain. For this reason, **crep** includes two tools which allow the user to either a) add rules on to the existing delimiter (**build_delim**), or b) construct a custom sentence delimiter from scratch (**build_delim_user**).

If the user is satisfied with the performance of the existing default delimiter, this section can be skipped. If, however, the default delimiter is found to be insufficient for particular needs, whether one wishes to have one's corpus delimited by regular grammatical sentences with domain-specific enhancements, or something different altogether (separated at semicolons only, for instance), then one may find this section helpful in using **crep** to achieve one's searching needs.

6.1 Examining the output from a delimiter

Examining output from a **crep** delimiter is a logical place to start in tutoring the user on **crep's** delimiting, as it would help the user in at least three ways:

a) in understanding how the **crep** default delimiter works;
b) in knowing how to test one's own custom delimiters; and
c) in using the output of the delimiters and the input to other tools besides **crep**.

The user should become familiar with what the constructed delimiters do to raw text. In other words, the user should become familiar with the text that the delimiters insert in the corpus to signal to `pos` and **crep** where the ends of sentences fall. Examine example 25 below which uses the default delimiter:

EXAMPLE 25.

```
$cat  story.txt
    Once upon a time, there was an aardvark named Vance.  He owned a
nice flat in the Lower East Side.  Unlike other aardvarks, he didn't
eat any ants.  Indian food was actually his favorite.
    One day, Vance put on his best three-piece suit in search of a
tasty lunch. When he was enjoying his Biryani, he noticed across the
dining room a very handsome frog, dressed businesslike, with the Wall
Street Journal under his arm.  Vance walked over to him and asked,
"What would you call behavior such as ours?"
    The frog dabbed his mouth daintily with the corner of his napkin
and returned, "When in Rome, do as the Romans do."

$
```

Now we input `story.txt` *into the default delimiter (called* **sent***). We will pass* **sent's** *output into the* **crep** *tool* **delim_export** *to make the output more readable (see Section 7.4 for more information on* **delim_export***).*

```
$sent  <  story.txt  |  delim_export  "
>     ***END!***
> "
    Once upon a time, there was an aardvark named Vance.
  ***END!***
He owned a nice flat in the Lower East Side.
  ***END!***
Unlike other aardvarks, he didn't eat any ants.
  ***END!***
Indian food was actually his favorite.

  ***END!***
One day, Vance put on his best three-piece suit in search of a tasty lunch.
```

```
  ***END!***
When he was enjoying his Biryani, he noticed across the dining room a very
handsome frog, dressed businesslike, with the Wall Street Journal under his
arm.
  ***END!***
Vance leaned over to him and asked, "What would you call behavior such as
ours?"

  ***END!***
The frog dabbed his mouth daintily with the corner of his napkin and
returned, "When in Rome, do as the Romans do."
  ***END!***
```

We should notice that the delimiter puts each new sentence on a new line.

6.2 Adding features to crep's delimiter

The default delimiter does not take into account such exceptions such as Mr., Mrs., and Dr. For example, the consider the corpus `doctor.txt`:

EXAMPLE 26.

```
$cat  doctor.txt
Mr. T. went to see Dr. Shoebuckle last week.  Shoebuckle is the only doctor
who can treat malcodosis (bites from software bugs) this side of Silicon
Valley.  In just three hours, Shoebuckle cured Mr. T. The next day, Mr. T.
notified Mrs. McFiggles at the Pentagon about the feat.
$
```

If we pass `doctor.txt` *into the default delimiter, we will get the following delimitations (the output format is the same as that of the corpus in Section 6.1):*

```
$sent  <  doctor.txt  |  delim_export  "
>     ***END!***
> "
Mr.
  ***END!***
T. went to see Dr.
  ***END!***
Shoebuckle last week.
  ***END!***
Shoebuckle is the only doctor  who can treat malcodosis (bites from
software bugs) this side of Silicon  Valley.
  ***END!***
In just three hours, Shoebuckle cured Mr.
  ***END!***
T.
```

```
  ***END!***
The next day, Mr.
  ***END!***
T.  notified Mrs.
  ***END!***
McFiggles at the Pentagon about the feat.
  ***END!***
$
```

Notice that the delimiter thought the titles Mr., Mrs., and Dr. signified the end of a sentence when followed by a proper name. We would like to add capabilities to the delimiter to handle these exceptions. To do so, we will use the **build_delim** tool. If we wished to replace the default delimiter entirely with a new one, we would use **build_delim_user**, which will be discussed in Section 6.3.

6.2.1 Introduction to the delimiter source file

The user articulates how to change the delimiter by creating a **delimiter source file**. There are three kinds of input one could specify in a delimiter source file to either build a new delimiter or add on to the existing default delimiter, all in `lex` syntax:

a) definitions (similar to the rule section of a `lex` source file);

b) rules for when a sentence should be broken off (called cut-rules);

c) rules for when a sentence should NOT be broken off (called not-rules).

To get an understanding of what these modifiables could be, let's take a look at items a), b), and c) in the default delimiter:

EXAMPLE 27.

Definitions:

```
Digits            [0-9]+
StuffBeforeDot    ([a-z]|[A-Z]|[\)])+
WON               ([ ]|[\n])+
PunctSep          [\?!;]
```

Cut-Rules:

```
{Digits}\.{WON}([A-Z]|\")
{StuffBeforeDot}\.\"{WON}([A-Z]|\")
{StuffBeforeDot}{PunctSep}(\"|\)\.)?{WON}([A-Z]|\")
```

```
{StuffBeforeDot}\.{WON}([A-Z]|\")
```

Not-rules:

{ There are none in the default delimiter }

These modifiables are actually embedded in a `lex` source file `sent.l` (the modifiables are boldfaced for clarity):

EXAMPLE 28.

```
/* Default Sentence delimiter          */
/* For use with crep package           */


/* the executable for this lex file will delimit sentences of the */
/* corpus specified as standard input by the line                 */
/* '\n.PP\n.End of Discourse\n' so POS can process it             */

Digits               [0-9]+
StuffBeforeDot        ([a-z]|[A-Z]|\))+
WON                  [ \t\n]+
PunctSep             [\?!;]


%%
{Digits}\.{WON}([A-Z]|\")                                   |
{StuffBeforeDot}\.\"{WON}([A-Z]|\")                         |
{StuffBeforeDot}{PunctSep}(\"|\)\.)?{WON}([A-Z]|\")         |
{StuffBeforeDot}\.{WON}([A-Z]|\")                           {
yyless(yyleng - 1);
                                                       ECHO;
                                                       printf("\n.End of
sentence
\n.PP\n.End of Discourse\n"); }

     /*     Not-rules section is empty.                 */
     /*     Action for not-rules is           ECHO;     */
\n                             |
\n[ ]                                printf(" ");

Codes:.*\n                           ;

<<EOF>>                                         {  ECHO; printf("\n\n\n.PP
\n.End of Discourse\n"); printf("eNDeNDeND.\n\n\n.PP\n.End of
Discourse\n"); yyterminate(); }
```

Since this file contains much more than the user needs to know about sentence delimiting, **crep** abstracts the the user's input in the creation of a sentence delimiter to just the three modifiables mentioned above -- the syntax of the delimiter source file for the above `lex` source file appears as follows:

EXAMPLE 29.

```
Digits             [0-9]+
StuffBeforeDot     ([a-z]|[A-Z]|\))+
WON                [ \t\n]+
PunctSep           [\?!;]
%%
{Digits}\.{WON}([A-Z]|\")                                  |
{StuffBeforeDot}\.\"{WON}([A-Z]|\")                        |
{StuffBeforeDot}{PunctSep}(\"|\)\.)?{WON}([A-Z]|\")        |
{StuffBeforeDot}\.{WON}([A-Z]|\")                          |
%%

%%
```

There are three sections in the delimiter source file: definitions, rules, and not-rules; each section is ended by a '%%' on its own line. The syntax is similar to `lex`; in fact, the abbreviations section is exactly as it would appear in a lex file; the rules and not-rules sections resemble `lex` rules *without* actions. There are no actions in a delimiter source file because they are inserted by **build_delim**. Note that each cut-rule in the cut-rules section is ended with a '|', even the last one.

For the default delimiter, there are no not-rules; hence, the not-rules section of the delimiter source file is blank. The '%%' which ends the section, however, must be left in. A not-rule will, however, be employed in the following example.

6.2.2 Creating a delimiter source file with the 'Titles' enhancement

As seen from example 26, the default delimiter handles titles such as Mr., Mrs., and Dr. improperly. Why is this so? Let's look at one of the `lex` cut-rules in the default delimiter (from Example 29) to see why:

```
{StuffBeforeDot}\.{WON}([A-Z]|\")
```

This cut-rule means in plain language that any string containing one or more letters, followed by a period, followed by whitespace, followed by a capital letter or double quote, signifies a sentence break. In fact, that capital letter or double quote is the first letter of the new sentence. In most sentences, this cut-rule works fine (the boldfacing is the match):

```
Hello there.   My name is Roggle.
```

Notice how this cut-rule thwarts a sentence with a title:

```
Hello, Mr.  Vindarten.
```

Now that we know why titles get chopped by the current delimiter, we can write a not-rule to circumvent this unfortunate match. First, we shall define our titles:

```
Title           (Mr\.)|(Mrs\.)|(Dr\.)|(Ms\.)
```

Next, we want to write a not-rule which will state when the delimiter should *not* delimit a sentence -- the case of Mr. Vindarten above. We shall reuse the whitespace definition WON (found in example 29):

```
{Title}{WON}[A-Z]   |
```

This is almost correct. The very cut-rule we are trying to circumvent is competing with this not-rule as both rules match the same number of characters. `lex`, by convention, picks the first rule in the file when two rules match exactly the same text -- not desirable behavior for our purposes; therefore, we must add one more look-ahead character to make the not-rule longer:

```
{Title}{WON}[A-Z](\.|{WON}|[a-z])      |
```

The last lookahead character could be either a whitespace, a lowercase letter, or a period, in the case of Mr. T. This lookahead character is sufficient to make this not-rule's matching strings longer than those of the thwarting cut-rule's matching strings. This lone not-rule will become the not-rules section of our delimiter source file.

We are not finished yet, however. There is at least one not-rule involving titles which should indeed indicate the end of sentence. Consider the following sentences from `doctor.txt`:

```
In just three hours, Shoebuckle cured Mr. T. The next day, Mr. T.
notified Mrs. McFiggles at the Pentagon about the feat.
```

Now we must explicitly tell the delimiter to terminate the sentence if a one-letter name ends a sentence:

```
{Title}{WON}[A-Z]\.{WON}([A-Z]|\")       |
```

We shall now integrate all three modifiables into one file and pass it in to **build_delim**.

6.2.3 Building the delimiter with the delimiter source file

Once the user has created the delimiter source file, he/she is ready to make a call to **build_delim** to construct the new delimiter. We'll use the source file we just created in section 6.2.2 (we named it `titles.sent`):

EXAMPLE 30.

```
Title           (Mr\.)|(Mrs\.)|(Dr\.)|(Ms\.)
%%
{Title}{WON}[A-Z]\.{WON}([A-Z]|\")       |
%%
{Title}{WON}[A-Z](\.|{WON}|[a-z])        |
%%
```

build_delim takes two required parameters: the name of the target executable, and the name of the delimiter source file:

EXAMPLE 31.

```
$build_delim  my_new_delim  titles.sent
Building new Sentence Delimiter based on user file titles.sent...
Done. Delimiter is saved in executable file my_new_delim.
Flex code is saved in my_new_delim.flex.
$
```

To check to see if our rules are correct, we shall input `doctor.txt` into `my_new_delim`:

EXAMPLE 32.

```
$my_new_delim  <  doctor.txt  |  delim_export  "
>     ***END!***
> "
Mr. T. went to see Dr. Shoebuckle last week.
  ***END!***
Shoebuckle is the only doctor  who can treat malcodosis (bites from
software bugs) this side of Silicon  Valley.
  ***END!***
In just three hours, Shoebuckle cured Mr. T.
  ***END!***
The next day, Mr. T.  notified Mrs. McFiggles at the Pentagon about the
feat.
  ***END!***
$
```

As we can see, we have achieved success with just one cut-rule and one not-rule. Compare this output with that from the default delimiter `sent` in example 26 (Section 6.2). As a general guideline, try to accomplish as much as possible in as few rules as possible. `lex` can match at most two rules on the same text fragment simultaneously[8] .

6.3 Using build_delim_user to create a delimiter containing no default rules, not-rules, or definitions

The user may find a need for having total control over writing all the rules for the delimiter, borrowing nothing from the default delimiter. To build such a delimiter, use **build_delim_user** instead of **build_delim**. It takes the same parameters as **build_delim**, but the resulting delimiter contains only the sentence delimiting information from the user's delimiter source file.

> ***A HELPFUL TIP***
>
> Building a delimiter with **build_delim_user** is a useful way of isolating the behavior of a lone cut-rule if you have only one cut-rule in your delimiter source file. Since the output of the delimiter in this case will reflect only delimiting from that one cut-rule, the user can then readily test the functionality of cut-rules on a rule-by-rule basis.

6.4 A note about read-ahead characters

All sentence delimiters created with **build_delim** or **build_delim_user** expect all cut-rules to have one read-ahead character (not-rules don't need them; cut-rules *require* them). If ones enter a cut-rule such as the following

```
[A-Za-z]\.          |
```

in the cut-rule section of a delimiter source file, the cut-rule will indeed match what it is supposed to, but being that the delimiter expects the cut-rule to have a read-ahead character, this cut-rule applied to a sentence would yield the following results:

[8] See *lex & yacc* (O'reilly and Associates 1992) for more information.

EXAMPLE 33.

Text from corpus:

```
He saw it coming.  He knew his houseplants would acquire posable thumbs
after reading Darwin aloud to them.
```

Output from delimiter with lone rule " `[A-Za-z]\.           |`*":*

```
He saw it coming
  ***END!***
.  He knew his houseplants would acquire posable thumbs after reading
Darwin aloud to them
  ***END!***
.
  ***END!***
```

The period, instead of ending a sentence, became the first character of the next sentence. To generalize, the last character matching the cut-rule always becomes the first letter of the next sentence. Make sure the cut-rules account for this fact.

6.5 Editing the `lex` source code file directly

For users who would prefer to edit the actual `lex` source file for the default delimiter directly without using the **build_delim** or **build_delim_user** interface may do so. The user must be careful, however, not to corrupt the actions of the cut-rules and not-rules; otherwise, the delimiter will not work properly. The path of the `lex` source code for the default delimiter, `sent.l`, appears in Appendix B.

6.6 Using delimited output for uses other than input to crep

The tools **build_delim** and **build_delim_user** are built to readily work with **crep**. One may, however, delimit sentences with these tools and use the output for other purposes -- say, as input to other corpus tools. Other tools will inevitably accept sentences delimited with different strings (i.e. not the 'End of sentence' strings which **crep** uses); the **crep** tool **delim_export** facilitates porting of **crep**-delimited text by allowing the user to easily convert the delimiting strings to something different. **delim_export** effectively eliminates the need to construct a file scanner to modify the delimited corpus. See Section 7.4 for a complete description. **delim_export** is also useful in making the output of a **crep** delimiter more readable.

6.7 Using custom delimiters with crep

crep can either use a pre-built custom delimiter or build a delimiter on the fly. We will first demonstrate how to use a pre-built custom delimiter. The next example includes the `-s` switch; the custom delimiter we built in example 31 is the parameter of `-s`:

EXAMPLE 34.

```
$cat  doctor.txt  |  crep  -e  'Mr'  -s  my_new_delim
crep: Using user-defined sentence delimiter my_new_delim.
Parsing Your Regular Expression(s)...
crep: Expression Parser Warning:
  Expression `Mr` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Tagging words in your corpus file(s)...
Building the flex file...
Regular Expression: Mr
Compiling the flex file...
Executing Flex File...

Matching sentences:
Mr. T. went to see Dr. Shoebuckle last week.

In just three hours , Shoebuckle cured Mr. T.

The next day , Mr. T.  notified Mrs. McFiggles at the Pentagon about the
feat.

$
```

A delimiter may also be built on the fly in an actual call to **crep** via `-b` and `-B`. `-b` is used to call **build_delim**; `-B` sends its parameter to **build_delim_user**:

EXAMPLE 35.

```
$cat  doctor.txt  |  crep  -e  'Mr'  -b  'my_new_delim  titles.sent'
Parsing Your Regular Expression(s)...
crep: Expression Parser Warning:
  Expression `Mr` was not found in your definition file.
  It will be inserted LITERALLY into the flex file.
Building new Sentence Delimiter based on user file titles.sent...
   Done. Delimiter is saved in executable file my_new_delim.
   Flex code is saved in this.flex.
Tagging words in your corpus file(s)...
```

```
Building the flex file...
Regular Expression: Mr
Compiling the flex file...
Executing Flex File...

Matching sentences:
Mr. T. went to see Dr. Shoebuckle last week.

In just three hours , Shoebuckle cured Mr. T.

The next day , Mr. T.  notified Mrs. McFiggles at the Pentagon about the
feat.

$
```

The output is, of course, the same as that of Example 34; the same delimiter was used, but in example 35 we built it in a call to **crep**. `-B` may be invoked in a similar fashion to make a call to **build_delim_user**.

Note that even in a call to **crep** with `-b` or `-B`, the executable delimiter as well as its corresponding `lex` source code files are saved just as they are in straight calls to **build_delim** and **build_delim_user**.

6.8 Nonconventional delimiters

build_delim and to a greater degree **build_delim_user** can be used to construct a delimiter to break up a corpus in just about any arbitrary way. Consider the following corpus:

EXAMPLE 36.

```
$cat  letter.txt
Dear Sirs stop I have made a decision regarding your offer for employment
stop By this time tomorrow I will be working in a fish processing factory
in Alaska stop I have found my calling stop I would have taken your offer
but it lacked adventure stop Sincerely Garman E. Kak stop
```

With the delimiter source file `stop.sent` *appearing as follows:*

```
$cat stop.sent
%%
stop[ ]      |
%%

%%
$build_delim_user  stopdel  stop.sent
```

```
Building new Sentence Delimiter based on user file stop.sent.
New executable delimiter will contain NO default rules or definitions...
   Done. Delimiter is saved in executable file stopdel.
   Flex code is saved in stopdel.flex.
$
```

We can now break up the corpus accordingly :

```
$stopdel  <  letter.txt  |  delim_export  "
>     ***END!***
>"
Dear Sirs stop
  ***END!***
 I have made a decision regarding your offer for employment stop
  ***END!***
  By this time tomorrow I will be working in a fish processing factory in
Alaska stop
  ***END!***
 I have found my calling stop
  ***END!***
 I would have taken your offer but it lacked adventure stop
  ***END!***
 Sincerely Garman E. Kak stop
  ***END!***
$
```

To generalize, the user may create a delimiter to break up a corpus in may different ways -- by paragraph, by semicolon, or by most any arbitrary delimitation, simply by changing the entries in a delimiter source file and running **build_delim** or **build_delim_user**.

6.9 Analyzing the created `lex` file for errors in your original delimiter source file

In every call to **build_delim** or **build_delim_user**, the `lex` file created from the delimiter source file is saved for the user's benefit (note the line '`Flex code is saved in <filename>.flex`' every time **build_delim** and **build_delim_user** are called). An analysis of the `lex` file may reveal errors inherited from the original delimiter source file. If, for instance, the `lex` file contains strange uncompilable code, the user may then reference the corresponding rule or rules in the delimiter source file to fix the error.

7. OTHER ASSOCIATED TOOLS OF crep

In addition to **build_delim** and **build_delim_user** (Section 6), the following five tools can be used in conjunction with **crep**. The first two, **crep_clean** and **crep_prep**, were designed exclusively for **crep**. The last three, **rcat**, **delim_export**, and **diff_clean**, may be used more universally.

7.1 crep_clean

After using the `-k` option, the user may wish to delete the temporary files directly, and not have to run **crep** again to destroy them. By running **crep_clean**, the **crep** temporary files in the current directory are removed one by one, asking the user for verification at each filename.

EXAMPLE 37.

```
$crep_clean
rm: remove EmptyExpFilecrep? y
rm: remove EmptyExpFilecrep.make? y
rm: remove EmptyExpFile.lex.crep? y
rm: remove EmptyExpFile? y
rm: remove in.tagged? y
rm: remove match_file? y
rm: remove sent_file? y
rm: remove crephelp.o? y
rm: remove Found_nums.txt? y
```

```
rm: remove lex.yy.o? y
rm: remove lex.yy.c? y
rm: remove HowManyExps? y
rm: remove TempExpStuff2? y
rm: remove TempExpStuff4? y
rm: remove good_exps? y
rm: remove raw_exps? y
rm: remove ExpNames? y
crep_clean: Cleanup done.
$
```

Peeking at some of these files may prove interesting to those who are interested in what makes **crep** tick.

7.2 crep_prep

If the user would prefer to tag many corpora in one stage rather than having to run **crep** each time he or she wants a new corpus tagged, **crep_prep** can be used. **crep_prep** simply tags and preps the input (from standard input) so it may be used with **crep**. **crep_prep** performs the tasks of the tagging module in figure 4-1.

crep_prep takes one required parameter (the name to give the file that will hold the tagged input) and one optional parameter (the name of a user-defined sentence delimiter to use instead of the default).

EXAMPLE 38.

```
$cat  sample  |  crep_prep  sample.tagged
The output is being sent to sample.tagged.
$
```

EXAMPLE 39.

```
$cat  sample  |  crep_prep  sample.tagged  my_delim
Using user-defined sentence delimiter my_delim.
The output is being sent to sample.tagged.
$
```

A HELPFUL TIP

When you pre-tag input, and then change the sentence delimiter, the pre-tagged input will always reflect the delimitations provided by the old delimiter, since delimiting occurs *before* tagging (checking for delimitation *after* tagging is unrealistic, since tags must also be taken into account). You must re-tag input to reflect the new delimiter. Section 6 covers building a new delimiter.

7.3 rcat

rcat is a general-purpose recursive `cat` function. **rcat** will `cat` every file in the directory (specified as its parameter), and then will recursively `cat` all the files in any the directory's subdirectories, and so on. Here are the shell functions:

```
function catall {
  for i in `/bin/ls`
   do
     echo ITEM: $i
     if [ -d "$i" ]; then
        cd $i
        catall $i
        cd ..
     else cat $i
     fi
   done
}


function real_rcat {
  cd $1
  catall
  cd $2
}

function rcat {
  real_rcat $1 $PWD/
}
```

rcat can be used to input textual corpora into **crep**, but can also be used with `grep`, or any UNIX executable that accepts input from standard input.

7.4 delim_export

Corpora delimited by **crep** may easily be inputted into other corpus tools with **delim_export**. Since tools may not agree on a standard sentence delimitation marker, **crep** allows the user to change the marker to any arbitrary string without needing to construct a file filter. Simply give **delim_export** the delimiting string as its only parameter, and pass in the delimiter corpus as standard input (See example 40).

Thus, the **build_delim** interface may be used to create a delimiter which is used with corpus tools other than **crep.** In example 40, we shall run **delim_export** on the corpus introduced in section 6.1. Keep in mind that below we use **sent**, the default delimiter; but the user may use *any* delimiter created with either **build_delim** or **build_delim_user** in conjunction with **delim_export**.

As an added advantage, **delim_export** can be utilized to improve the clarity of delimited output when intended for human consumption.

EXAMPLE 40.

```
$cat  story.txt  |  sent  |  delim_export  "
>     ***END!***
>  "
    Once upon a time, there was an aardvark named Vance.
  ***END!***
He owned a nice flat in the Lower East Side.
  ***END!***
Unlike other aardvarks, he didn't eat any ants.
  ***END!***
Indian food was actually his favorite.
  ***END!***
One day, Vance put on his best three-piece suit in search of a tasty lunch.
  ***END!***
When he was enjoying his Biryani, he noticed across the dining room a very
handsome frog, dressed businesslike, with the Wall Street Journal under his
arm.
  ***END!***
Vance leaned over to him and asked, "What would you call behavior such as
ours?"
  ***END!***
The frog dabbed his mouth daintily with the corner of his napkin and
returned, "When in Rome, do as the Romans do."
  ***END!***
```

7.5 diff_clean

diff_clean removes the <, >, and number codes output by `diff`. **diff_clean** may be used in conjunction with **crep** or outside of **crep** when the `diff` formatting characters do not wish to be seen. An example using **diff_clean** appears in Example 17.

APPENDIX A. SUMMARY OF crep OPTIONS

-b <'delim source abb_file'>	Builds a sentence delimiter to be used in place of the default delimiter. User's rules in source and abb_file add on to default rules
-B <'delim source abb_file'>	Builds a sentence delimiter to be used in place of the default delimiter. User's rules in source and abb_file replace all rules
-c <infile>	uses <infile> as input instead of standard input
-d <exp_definition_file>	uses a file <exp_definition_file> containing **crep** expression variables
-D <exp_definition_file>	uses a file <exp_definition_file> containing lex expression variables
-e <regular_expression>	specifies a regular expression parameter in **crep** syntax on the command line.
-E <regular_expression>	specifies a regular expression parameter in lex syntax on the command line.
-f <crep_reg_exp_file>	specifies a file containing a regular expression parameter in **crep** syntax.

`-F <crep_reg_exp_file>` specifies a file containing a regular expression parameter in `lex` syntax.

`-g <mexp_file>` Reads in a `-m` parameter from file `mexp_file`.

`-k` keeps temporary files which are created in the current directory. The temporary files can then be used by -x to speed up execution time.

`-m '<exp1 file1 exp2 file2 ...>'` specifies multiple regular expressions and redirects their respective matches to different output files. With this option, each expression E_i must be immediately followed by a filename F_i. The corpus sentences matching each E_i are copied to the file of path F_i.

`-n <tagged_filename>` specifies a name in which to save a tagged version of free text input. The file created by -n will not be automatically destroyed like all other temporary files (unless `-k` is used, in which case ALL temporary files as well as the file created by -n are saved). `tagged_filename` can then be inputted into **crep** using the `-p` or `-P` options.

`-p` tells **crep** that the standard input is already tagged. **crep** will skip the tagging and just pass the pre-tagged input into the execution of the `lex` file.

`-P <infile>` is like `-c`, but specifies a TAGGED corpus. Like `-p`, **crep** is instructed to skip the tagging.

`-s <delimiter>` Use `delimiter` to delimit sentences rather than the default delimiter.

`-t` outputs tagged matching sentences. By default **crep** strips off the tags.

`-w` includes what phrases matched what expression in the sentence. The matching phrase is displayed under the matching sentence in the form

`<expression>: <matching_phrase>`

By default, **crep** simply outputs the matching sentences without this information.

`-x` speeds up execution by skipping the compilation time normally needed by **crep**. NOTE: this option can only be used after the `-k` option has been used. Also, since `-x` uses the previously compiled `lex` file, if the user changes either the expression parameter or the `-d/-D` parameter, a new `lex` file must be compiled. `-x` is only useful if the user wishes to run the *same* expression with *different* input files or with *different* output formats.

APPENDIX B. WHERE TO FIND crep

crep and all of the associated tools (with the exception of `pos`) are located in the Columbia University CS file system in `/u/peptic/darrin/crep/bin/`.

The test corpora `sample`, `doctor.txt`, and `story.txt` live in `/u/peptic/darrin/corpus/`.

`sent.l` appears in `/u/peptic/darrin/crep/src/tag_mod/`.

A more detailed version of the variable definition file `ExpFile` is located in `/u/peptic/darrin/lexp/`.

rcat lives in `/u/peptic/darrin/crep/bin/` as `rcat_code`. The user may include **rcat** in his/her `.profile`, or any executable shell script.

The source code for `pos` lives in `/u/rhythmics/smadja/POS/`. A shell script which calls the tools of `pos` appears in `/u/peptic/darrin/crep/bin/`. For more information on `pos`, the user should contact Ken Church at `kwc@research.att.com`.

If the user is accessing a copy of **crep** which lives somewhere besides the Columbia University CS file system, or if any of the above files seem to be moved, check the `readme` file in the **crep** directory for the most up-to-date information on **crep**.

APPENDIX C. ERROR MESSAGES

There are two kinds of errors the user may see: **crep's** own error messages, and those of the external tools (`lex` and `pos`) called by **crep**.

C.1 crep's own error messages

As a convention, any error caught directly by **crep**, and not one of the other tools, will be prefaced by '`crep:`' in the output. Warnings will be prefaced by '`crep: <module> warning:`'. Warnings may not be harmful to the functionality of **crep**, and in fact may be the desirable behavior of the user.

C.2 Errors signaled by `lex`

If there are `lex` syntax errors a) in the variable definition file; b) the regular expression; or c) the user-authored sentence delimiter file (Section 6), `lex` will signal an error when compiling from within **crep**, and **crep** will abort execution. The `lex` documentation should be referenced for further information on these errors.

C.3 Errors signaled by `pos`

Any errors trapped while in the tagging module not prefaced by **crep**: will be `pos` errors.

APPENDIX D. LISTING OF `pos` TAGS

Following is a listing of the part of speech tags assigned by `pos`. The tag set is identical to the set used to tag the Brown Corpus in Francis and Kucera's *Frequency Analysis of English Usage: Lexicon & Grammar* (Houghton Mifflin, 1982).

```
.                            end of sentence
NN                           singular noun
IN                           preposition
AT                           article
NP                           proper noun
JJ                           adjective

NNS                          plural noun
CC                           conjunction
RB                           adverb
VB                           un-inflected verb
                                (infinitive form or present and not 3rd
                                 person singular)
VBN                          verb +en
                                (e.g., taken, looked (passive, perfect))
VBD                          verb +ed (e.g., took, looked (past tense))
CS                           subordinating conjunction
PPS                          subject pronoun
VBG                          verb +ing
PP$                          possessive pronoun
CD                           number
PPSS                         pronoun
TO                           the word 'to' as an infinitive marker
```

MD	modal
PPO	object pronoun
BEZ	is
BEDZ	was
AP	
DT	demonstrative
''	
``	
QL	qualifier
VBZ	verb +s (3rd person singular)
BE	un-inflected form of 'to be'
RP	particle
WDT	wh
HVD	had
$	's (as a possesive)
BER	were
*	negation
WRB	wh adverb
HV	have
WPS	who (as a subject)
BED	were
ABN	
DTI	
PN	someone
HVZ	has
BEN	been
)	
DTS	
(	
EX	there
NR	adverbial noun phrase
DO	do
:	
DOD	did
PPL	
DTX	
BEG	being
UH	interjection
DOZ	does
BEM	am
ABL	
PPLS	reflexive pronoun
WPO	who (as an object)
HVG	having
QLP	
WP$	whose
HVN	had (+en)
WQL	
ILLEGAL	should not be used