Chapter 3

Some Initial Processing

Now that we have learned some things about Humdrum representations (and the `**kern` representation in particular), let's explore some basic processing tasks.

The *census* Command

The Humdrum **census** command provides basic information about an input stream or file. We can invoke the command by typing the command-name followed by the name of a file. The command:

```
census india01.krn
```

might produce the following output:

```
HUMDRUM DATA

Number of data tokens:      91
Number of null tokens:      0
Number of multiple-stops:   0
Number of data records:     91
Number of comments:         14
Number of interpretations:  7
Number of records:          112
```

Most commands provide *options* that will modify the operation of the command in a particular way. In UNIX-style commands, options follow after the command-name and are typically specified by a single letter preceded by a hypen. The **-k** option with the **census** command will give further information pertaining to the Humdrum `**kern` representation. With the **-k** option, the output includes the number of notes in the file, the longest, shortest, highest, and lowest notes, the maximum number of concurrent notes or voices, the number of rests, and the number of barlines. For example, the command:

```
census -k india01.krn
```

might produce the following *additional* output:

```
KERN DATA

Number of noteheads:        78
Number of notes:            78
Longest note:               1
Shortest note:              16
Highest note:               cc
Lowest note:                c
Number of rests:            1
Maximum number of voices:   1
Number of single barlines: 11
Number of double barlines: 1
```

Notice that a distinction is made between the number of notes and the number of noteheads. A tied note is considered to be a single "note," although it may be notated using two or more noteheads.

The output from **census** can be restricted to a particular item of information by "piping" the output to the UNIX **grep** command.

Simple Searches using the *grep* Command

The UNIX **grep** command is a popular tool for searching for lines that match some specified pattern. Patterns may be simple strings of characters, or may be more complicated constructions defined using the UNIX *regular expression* syntax. Regular expressions will be described in detail in Chapter 9. The command-name "`grep`" is an acronym for "get regular expression."

Useful patterns are often literal character strings, such as keywords. For example, the following command identifies whether the file `opus28.krn` contains the word "`Andante`":

```
grep 'Andante' opus28.krn
```

Every line containing the specified pattern will be output. If no match is found, no output is given.

Using a single command, all files in the current directory can be searched by substituting the asterisk (shell *wildcard*) in place of the filename. The following command identifies all instances where the word "`Andante`" occurs; all files in the current directory are searched:

```
grep 'Andante' *
```

Once again, every line containing the sought pattern is echoed in the output. If more than one pattern is found, each instance of the pattern will be output on a separate line. Whenever a "wildcard" is used as part of the filename, **grep** causes the *name* of each file to be prepended to the output for all patterns that are found:

```
opus28:!! Andante
opus29:!! Andante
opus46:!! Andante
```

```
opus91:!! Andante
opus98:!! Andante
```

By default, **grep** distinguishes upper- and lower-case characters, so the above command will not match strings such as "ANDANTE". However, the **-i** option tells **grep** to ignore the case when searching. E.g.,

```
grep -i 'Andante' *
```

Sought patterns may occur in any line, including data records and comments. The following command will identify the presence of any double-sharps in the file `schumann.krn`.

```
grep '##' schumann.krn
```

Pattern Locations Using *grep -n*

If a pattern is found, it is sometimes helpful to know the precise location of the pattern. The **-n** option tells **grep** to prepend the *line number* for each matching instance. The following command identifies the line numbers for lines containing a double sharp for the file `melody.krn`:

```
grep -n '##' melody.krn
```

The output might look like this:

```
1109:{4g##
1731:16g##
3002:16f##
```

— meaning that double sharps were found in lines 1109, 1731, and 3002 in the file `melody.krn`.

Counting Pattern Occurrences Using *grep -c*

In some cases, the user is interested in counting the total number of instances of a found pattern. The **-c** option causes **grep** to output a numerical *count* of the number of lines containing matching instances. For example, in the `**kern` representation, the beginning of each phrase is marked by the presence of an open curly brace ('{'). So the following command can be used to count the number of phrases in the file `glazunov.krn`:

```
grep -c '{' glazunov.krn
```

As noted, the **grep** command will search all lines (including comments) for matching instances of the specified pattern. If a curly brace were to appear in a comment or other non-data record, then our phrase-count would be incorrect. More carefully constructed patterns require a better knowledge of *regular expressions*. Regular expressions are discussed in Chapter 9.

Searching for Reference Information

As we saw in Chapter 2, Humdrum files typically encode library-type information using reference records. For example, the composer's name is encoded in a `!!!COM:` record, and the title is encoded via the `!!!OTL:` record. In conjunction with the **grep** command, these three letter codes provide useful tags to search for pertinent information. For example, the following command will identify the composer for the file `opus24.krn`:

```
grep '!!!COM:' opus24.krn
```

The output might look like this:

```
!!!COM: Boulanger, Nadia
```

Once again, wildcards (i.e., the asterisk) can be used to address all of the files in the current directory. Hence the command:

```
grep '!!!COM:' *
```

will produce a list of all composers of files in the current directory. Similarly, the following command will generate a list of all of the titles:

```
grep '!!!OTL:' *
```

The output might look as follows:

```
foster11:!!!OTL: Oh! Susanna
foster12:!!!OTL: Jeanie with the Light Brown Hair
foster13:!!!OTL: Beautiful Dreamer
foster14:!!!OTL: Gwine to Run All Night (or 'De Camptown Race')
foster15:!!!OTL: My Old Kentucky Home, Good-Night
foster16:!!!OTL: We are Coming, Father Abraam
foster17:!!!OTL: Don't Bet Your Money on De Shanghai
foster18:!!!OTL: Gentle Annie
foster19:!!!OTL: If You've Only Got a Moustache
foster20:!!!OTL: Maggie by my Side
foster21:!!!OTL: Old Folks at Home
foster22:!!!OTL: Better Times are Coming
foster23:!!!OTL: When this Dreadful War is Ended
foster24:!!!OTL: Hard Times Comes Again No More
```

Remember that when a wildcard is used in filenames, **grep** prepends the filename prior to found patterns. These filename 'headers' can be eliminated by selecting the **-h** option for **grep**:

```
grep -h '!!!OTL:' *
```

(N.B. Some older versions of **grep** do not support all of the options described here. Filename headers can be stripped from the output by using the UNIX **sed** described in Chapter 14.)

We might place the resulting list of titles in a separate file using the UNIX *file redirection* construction. The output of a command can be placed into a file by following the command with a greater-than sign (>) followed by a filename. For example, the following command places the output from **grep** in a file called `titles`:

```
grep -h '!!!OTL:' * > titles
```

Beware that if the file `titles` already exists then it will be over-written and its previous contents lost. With the **-h** option the file `titles` might contain the following lines:

```
!!!OTL: Oh! Susanna
!!!OTL: Jeanie with the Light Brown Hair
!!!OTL: Beautiful Dreamer
!!!OTL: Gwine to Run All Night (or 'De Camptown Race')
!!!OTL: My Old Kentucky Home, Good-Night
!!!OTL: We are Coming, Father Abraam
!!!OTL: Don't Bet Your Money on De Shanghai
!!!OTL: Gentle Annie
!!!OTL: If You've Only Got a Moustache
!!!OTL: Maggie by my Side
!!!OTL: Old Folks at Home
!!!OTL: Better Times are Coming
!!!OTL: When this Dreadful War is Ended
!!!OTL: Hard Times Comes Again No More
```

The *sort* Command

The UNIX operating system provides a general sorting utility called **sort**. We might use this utility to rearrange the titles in alphabetical order:

```
sort titles
```

Rather than using an intermediate file, we can directly connect the **grep** and **sort** commands using a UNIX "pipe." The vertical bar (|) creates a connection between the output of one command and the input of the next command. We can combine the above two commands to create an alphabetical listing of all titles in the current directory:

```
grep '!!!OTL:' * | sort
```

File-redirection can be added at the end of a pipe so the final output is captured in a file. In the follow case, the alphabetized titles are placed in the file `titles`:

```
grep '!!!OTL:' * | sort > titles
```

The *uniq* Command

Bach often harmonized a chorale melody more than once. In the 185 chorales in the original 1784 edition, several duplicate titles are present. Suppose you want to create an alphabetical list of titles

— but you want to exclude duplicate titles.

The UNIX **uniq** command provides a useful utility for eliminating duplication. Without any option, **uniq** simply eliminates any successive repeated lines. For example, given the input:

```
1
1
1
2
2
3
```

the **uniq** command will produce the following output:

```
1
2
3
```

Note that **uniq** only discards *successive* repeated records; an input such as the following would remain unmodified by the **uniq** command:

```
1
2
3
1
3
1
```

Another important point about **uniq** is that successive lines must be *exact repetitions* in order to be discarded. For example, if one line has a trailing blank that is not present in the previous line, then the line is not discarded.

Returning to our problem of creating a list of unique titles for J.S. Bach's chorale harmonizations, we can use the following command pipeline.

```
grep -h '!!!OTL:' * | sort | uniq
```

Note that our "pipeline" consists of three successive commands with the outputs connected to the inputs using the UNIX pipe symbol (|). The **sort** command is essential in order to collect identical titles as successive lines before passing the list to **uniq**.

Suppose you wanted to ensure that all of the works in the current directory are composed by the same composer. The same command structure can be used, only we would search for reference records encoding the composer's name:

```
grep -h '!!!COM:' * | sort | uniq
```

Even if the current directory contains hundreds of works by one composer (say Beethoven) and just a single work by another composer, the presence of the odd score will be obvious without hav-

ing to look through long lists:

```
!!!COM: Beethoven, Ludwig van
!!!COM: Stamitz, Carl Philipp
```

Of course we can make similar lists for other types of information available in reference records. The AIN reference record encodes instrumentation. We could make a list of various instrumental combinations used for scores in the current directory:

```
grep -h '!!!AIN:' * | sort | uniq
```

Options for the *uniq* Command

Like **grep**, the **uniq** command provides several options that modify its behavior. The **-d** option causes only those records to be output which are *duplicated* (i.e. two or more instances). Conversely, the **-u** option causes only those records to be output that are truly *unique* (i.e. only a single instance is present in the input).

Suppose, for example, that we want to know which of the Bach chorales are harmonizations of the same tunes — that is, have the same titles. (Of course the same chorale might be known by two or more titles, but let's defer this problem until Chapter 25.) The **-d** option will only output the duplicate records:

```
grep -h '!!!OTL:' * | sort | uniq -d
```

The output will identify those titles which appear in two or more files in the current directory. The output might look as follows:

```
!!!OTL: Befiehl du deine Wege
!!!OTL: Christ lag in Todesbanden
!!!OTL: Christus, der ist mein Leben
!!!OTL: Das alte Jahr vergangen ist
!!!OTL: Ein' feste Burg ist unser Gott
!!!OTL: Erbarm' dich mein, o Herre Gott
!!!OTL: Herr, ich habe missgehandelt
!!!OTL: Herr, wie du willst, so schick's mit mir
!!!OTL: Ich dank' dir, lieber Herre
!!!OTL: Jesu, der du meine Seele
!!!OTL: Jesu, meiner Seelen Wonne
```

Having established which titles are duplicates, a logical next step might be to identify the specific files involved. We can use **grep** again to search for a specific title. Without the **-h** option, the output will identify the appropriate filenames. For example:

```
grep '!!!OTL: Befiehl du deine Wege' *
```

might produce the following output:

```
bwv270.krn:!!!OTL: Befiehl du deine Wege
bwv271.krn:!!!OTL: Befiehl du deine Wege
bwv272.krn:!!!OTL: Befiehl du deine Wege
```

Sometimes we would like to have an output that contains *only* the *filenames* containing the sought pattern. The **-l** option causes **grep** to output only filenames that contain one or more instances of the sought pattern:

```
grep -l '!!!OTL: Befiehl du deine Wege' *
```

The output would appear as follows:

```
bwv270.krn
bwv271.krn
bwv272.krn
```

The **-u** option for **uniq** causes only unique entries in a list to be passed to the output. This is often useful in identifying works that differ in some way from other works in a group or corpus. For example, in some repertory, you may remember that a particular work had a different instrumentation than the other works. But you may not be able to remember what the specific instrumentation was. Use the **-u** option for **uniq** to produce a list consisting of only those works whose instrumentation differs from all others:

```
grep -h '!!!AIN:' * | sort | uniq -u
```

As in the case of the **grep** command, **uniq** also supports a **-c** option which counts the number of occurrences of a pattern. For example, if we want to count the number of works by each composer in the current directory:

```
grep -h '!!!OTL:' * | sort | uniq -c
```

The output might appear as follows:

```
 9 !!!COM: Berardi, Angelo
 2 !!!COM: Caldara, Antonio
12 !!!COM: Zarlino, Gioseffo
 2 !!!COM: Sweelinck, Jan Pieterszoon
 4 !!!COM: Josquin Des Pres
```

Notice that the number of instances is prepended to the reference records.

Incidentally, if we wanted to rearrange this list in order of the number of works, we could pass the above output to yet another **sort** command. Since **sort** sorts from left to right, it will begin sorting according to the numerical values at the extreme left. The command

```
grep -h '!!!OTL:' * | sort | uniq -c | sort
```

will rearrange the above output as follows:

```
  2 !!!COM: Caldara, Antonio
  2 !!!COM: Sweelinck, Jan Pieterszoon
  4 !!!COM: Josquin Des Pres
  9 !!!COM: Berardi, Angelo
 12 !!!COM: Zarlino, Gioseffo
```

It is important to understand that the two **sort** commands in our pipeline achieve different goals but use the same process. The first **sort** command sorts the composer's names into alphabetical order. This is done so that the ensuing **uniq** command is able to count successive identical records. Since the **uniq -c** command prepends numerical counts, the subsequent **sort** sorts first according to the numbers to the left of the reference records.

As a final note, we might mention that, like **grep** and **uniq**, the **sort** command has several options. One option, the **-r** option, causes the output to be arranged in reverse order. This can be useful in producing lists that are ordered from the most-common to the least-common.

Reprise

In this chapter we have introduced some elementary ways of processing Humdrum files. We noted that the **census** command can be used to identify basic statistics about a file. The **-k** option for **census** provides basic information related to `**kern` files — such as the number of notes and rests, the highest and lowest notes, the number of barlines, etc.

In this chapter we also introduced simple searching techniques using the **grep** command; **grep** provides a useful way of locating particular patterns of text characters in files. We used **grep** to identify composers, titles, instrumentation and other information. Most of our examples were limited to searching for Humdrum reference records. In later chapters we will use **grep** in more sophisticated searches. We noted several useful options for **grep**: the **-c** option causes a count to be output of the number of instances of the pattern in each file. The **-i** option causes **grep** to ignore any distinction between upper- and lower-case characters when searching for patterns. The **-h** option causes **grep** to avoid outputting the filenames prior to found patterns when more than one file is searched. The **-l** option results in only the filenames being output. In a later chapter we will encounter a number of other useful options provided by **grep**.

Also discussed in this chapter was the **uniq** command; **uniq** provides a useful utility for eliminating or isolating duplicate records or lines. Once again a number of useful options were introduced. The **-c** option causes **uniq** to prepend a count of the number of duplicate input lines. The **-d** option results in only duplicate input lines being noted in the output. The **-u** option does the reverse: only those input lines that are unique are passed to the output.

Finally, we introduced the UNIX **sort** utility. This command rearranges the order of successive input lines so they are in alphabetic/numeric order. The **sort** command provides a wealth of useful options; however, we mentioned only the **-r** option — which causes the output to be sorted in reverse order.