Chapter 12

Selecting Musical Parts and Passages

A Humdrum file may contain an encoding of a full score, or even of a large collection of scores. Often, we would like to isolate or extract particular parts or segments from a file or input stream. For example, we might want to extract a particular instrumental part, or select a group of related instruments; we might want to isolate a particular passage, remove certain measures, or extract a specific phrase; we might want to pull out a labelled section of a score (such as a *Coda* or *Da Capo*), or we might want to select a particular verse in a strophic song. In addition, we might want to isolate specific types of information, such as the figured bass, melodic interval information, or vocal text.

In this chapter we will explore two Humdrum tools for extracting material: **extract** and **yank**.

We know that Humdrum representations are structured like a grid with horizontal data ("records") representing concurrent information, and vertical data ("spines") representing sequentially occuring information. The Humdrum **extract** command can be used to isolate columns or spines of information. The **yank** command can be used to isolate rows or records. The **extract** command can be used to extract musical parts or other types of information that might be represented in individual spines. The **yank** command can be used to isolate passages or segments from an input, such as specified measures, phrases, or sections.

Extracting Spines: The *extract* Command

The **extract** command allows the user to select one or more spines from a Humdrum input. The command is typically used to extract parts (such as a tuba part) from some multi-part score. However, **extract** can also be used to isolate dynamic markings, musical lyrics, or any other stream of information that has been encoded as a separate Humdrum spine.

The **extract** command has several modes of operation. With the **-f** option, the user may specify a given data column (spine) or "field" to extract. Consider the opening of Bach's second Brandenburg Concerto shown in Example 12.1.

Example 12.1. J.S. Bach *Brandenburg Concerto No. 2*, mov. 1.

```
!!!COM: Bach, Johann Sebastian
!!!OPR: Six Concerts Avec plusieurs ... le prince regnant d'Anhalt-Coethen
!!!OTL: Brandenburgische Konzerte F
!!!XEN: Brandenburg Concerto No. 2 in F major.
!!!OMV: Movement 1.
!!!SCT: BWV 1047
!! [Allegro]
**kern   **kern   **kern   **kern   **kern   **kern   **kern   **kern   **kern   **kern
*ICklav  *ICstr   *ICstr   *ICstr   *ICstr   *ICstr   *ICstr   *ICww    *ICww    *ICbras
*Icemba  *Icello  *Icbass  *Iviola  *Ivioln  *Ivioln  *Ivioln  *Ioboe   *Ifltds  *Itromp
*IGcont  *IGcont  *        *        *        *        *        *        *        *
*IGripn  *IGripn  *IGripn  *IGripn  *IGripn  *IGripn  *IGconc  *IGconc  *IGconc  *IGconc
!cembal  !'cello  !Bd'rip  !Vd'rip  !v'lin2  !v'lin1  !v'lino  !oboe    !flauto  !tromba
*k[b-]   *k[b-]   *k[b-]   *k[b-]   *k[b-]   *k[b-]   *k[b-]   *k[b-]   *k[b-]   *k[]
*F:      *F:      *F:      *F:      *F:      *F:      *F:      *F:      *F:      *F:
*M2/2    *M2/2    *M2/2    *M2/2    *M2/2    *M2/2    *M2/2    *M2/2    *M2/2    *M2/2
*MM54    *MM54    *MM54    *MM54    *MM54    *MM54    *MM54    *MM54    *MM54    *MM54
*clefF4  *clefF4  *clefF4  *clefC3  *clefG2  *clefG2  *clefG2  *clefG2  *clefG2  *clefG2
8FF/     8FF/     8FFF/    8a\      8cc\     8ff\     8ff\     8ff\     8ff\     8f/
=1       =1       =1       =1       =1       =1       =1       =1       =1       =1
16F\LL   16F\LL   16FF\LL  8f\L     8a/L     8cc\L    8cc\L    8cc\L    8cc\L    8a/L
16G\     16G\     16GG\    .        .        .        .        .        .        .
16A\     16A\     16AA\    8c\      16f/LL   16a\LL   16a\LL   16a\LL   16a\LL   8cc/
16G\JJ   16G\JJ   16GG\JJ  .        16g/JJ   16b-\JJ  16b-\JJ  16b-\JJ  16b-\JJ  .
16F\LL   16F\LL   16FF\LL  8f\      8a/L     8cc\L    8cc\L    8cc\L    8cc\L    8a/
16G\     16G\     16GG\    .        .        .        .        .        .        .
16A\     16A\     16AA\    8c\J     16f/LL   16a\LL   16a\LL   16a\LL   16a\LL   8f/J
16G\JJ   16G\JJ   16GG\JJ  .        16g/JJ   16b-\JJ  16b-\JJ  16b-\JJ  16b-\JJ  .
16F\LL   16F\LL   16FF\LL  8f\L     8a/L     8cc\L    8cc\L    8cc\L    8cc\L    8a/L
16E\     16E\     16EE\    .        .        .        .        .        .        .
16F\     16F\     16FF\    8a\      8cc/     8ff\     8ff\     8ff\     8ff\     8cc/
16G\JJ   16G\JJ   16GG\JJ  .        .        .        .        .        .        .
16A\LL   16A\LL   16AA\LL  8cc\     8f/      8cc\     8cc\     8cc\     8cc\     8ff/
16B-\    16B-\    16BB-\   .        .        .        .        .        .        .
16A\     16A\     16AA\    8c\J     8cc/J    8ff\J    8ff\J    8ff\J    8ff\J    8cc/J
16G\JJ   16G\JJ   16GG\JJ  .        .        .        .        .        .        .
=2       =2       =2       =2       =2       =2       =2       =2       =2       =2
*-       *-       *-       *-       *-       *-       *-       *-       *-       *-
```

Suppose we wanted to extract the 'cello part. In the above encoding, the 'cello occupies the second spine (second field) from the left, hence:

```
extract -f 2 brandenburg2.krn
```

The resulting output would begin as follows:

```
!!!COM: Bach, Johann Sebastian
!!!OPR: Six Concerts Avec plusieurs ... le prince regnant d'Anhalt-Coethen
!!!OTL: Brandenburgische Konzerte F
!!!XEN: Brandenburg Concerto No. 2 in F major.
!!!OMV: Movement 1.
!!!SCT: BWV 1047
!! [Allegro]
**kern
*ICstr
*Icello
*IGcont
*IGripn
!'cello
*k[b-]
*F:
*M2/2
*MM54
*clefF4
8FF/
=1
16F\LL
16G\
etc.
```

Notice that the **extract** command outputs all global comments. In the case of local comments, **extract** outputs only those local comments that belong to the output spine.

The oboe and flauto dolce parts are encoded in spines 8 and 9. So we could extract the 'cello, oboe and flauto dolce parts by submitting a list of the corresponding fields. Spine numbers are separated by commas:

```
    extract -f 2,8,9 brandenburg2.krn
```

Numerical **ranges** can be specified using the dash. For example, if we wanted to extract all of the string parts (spines 2 through 7):

```
    extract -f 2-7 brandenburg2.krn
```

With the **-f** option, field specifications may also be made with respect to the right-most field. The dollars-sign character (`$`) refers to the right-most field in the input. The trumpet part can be extracted as follows:

```
    extract -f '$' brandenburg2.krn
```

(Notice the use of the single quotes to ensure that the shell doesn't misinterpret the dollar sign.) Simple arithmetic expressions are also permitted; for example '`$-1`' refers to the right-most field minus one, etc. By way of example, the command

```
    extract -f '$-2' brandenburg2.krn
```

will extract the oboe part.

Extraction by Interpretation

Typically, it is inconvenient to have to determine the numerical position of various spines in order to extract them. With the **-i** option, **extract** outputs all spines containing a specified *interpretation.* Suppose we had a file containing a Schubert song, including vocal score, piano accompaniment and vocal text (encoded using `**text`). The vocal text from the file `lieder` can be extracted as follows:

```
    extract -i '**text' lieder
```

(Notice again the need for single quotes in order to avoid the asterisk being interpreted by the shell.) Several different types of data can be extracted simultaneously. For example:

```
    extract -i '**semits,**MIDI' hildegard
```

will extract all spines in the file `hildegard` containing `**semits` or `**MIDI` data.

An important use of the **-i** option for **extract** is to ensure that a particular input contains only a specified type of information. For example, the lower-case letter '`r`' represents a rest in the `**kern` representation. If we wish to determine which sonorities contain rests, we might want to use **grep** to search for this letter. However, the input might contain other Humdrum interpretations (such as `**text`) where the presence of the letter '`r`' does not signify a rest. We can ensure that our search is limited to `**kern` data by using the **extract** command:

```
    extract -i '**kern' | grep  ...
```

Both exclusive interpretations and tandem interpretations can be specified with the **-i** option. For example, the following command will extract any *transposing* instruments in the score `albeniz`:

```
    extract -i '*ITr' albeniz
```

Tandem interpretations are commonly used to designate instrument classes and groups, so different configurations of instruments are easily extracted. The Brandenburg Concerto shown in Example 12.1 illustrates a number of tandem interpretations related to instrumentation classes and groups. For example, the interpretation `*ICww` identifies woodwind instruments; `*ICbras` identifies brass instruments; `*ICstr` identifies string instruments. In addition, `*IGcont` identifies "continuo" instruments; `*IGripn` identifies "ripieno" instruments; and `*IGconc` identifies "concertino" instruments. The following three commands extract (1) the woodwind instruments, (2) the ripieno instruments, and (3) any vocal parts, respectively.

```
    extract -i '*ICww' concerto4
    extract -i '*IGrip' brandenburg2
    extract -i '*ICvox' symphony9
```

Once again, more than one interpretation can be extracted simultaneously. The following command will extract the instrument-class "strings" and the instrument "oboe" from the file `milhaud`.

```
    extract -i '*ICstr,*Ioboe' milhaud
```

Similarly, the following command will extract the shamisen and shakuhachi parts from a score:

```
extract -i '*Ishami,*Ishaku' hito.uta
```

The behavior of **extract** is subtly different for tandem interpretations versus exclusive interpretations. Remember that exclusive interpretations are mutually exclusive, whereas tandem interpretations are not. Consider the following Humdrum representation:

```
**foo
a
b
c
**bar
x
y
z
*-
```

The command

```
extract -i '**foo'
```

will result in the output:

```
**foo
a
b
c
*-
```

Whereas the command

```
extract -i '**bar'
```

will result in the output:

```
**bar
x
y
z
*-
```

The `**foo` and `**bar` data are mutually exclusive. Now consider an input file where `foo` and `bar` are tandem interpretations:

```
**foobar
*foo
a
b
c
*bar
```

```
    x
    y
    z
    *-
```

The command

```
    extract -i '*foo'
```

will result in the output:

```
    **foobar
    *foo
    a
    b
    c
    *bar
    x
    y
    *-
```

Whereas the command

```
    extract -i '*bar'
```

will result in the output:

```
    **foobar
    *foo
    *bar
    x
    y
    z
    *-
```

When searching for a particular exclusive interpretation, **extract** resets each time a new exclusive interpretation is encountered. By contrast, when **extract** finds a target tandem interpretation, it begins outputting and doesn't stop until the spine is terminated.

Using *extract* in Pipelines

Of course the output from **extract** can be used to generate inputs for other Humdrum tools. Here are a few examples.

Recall that the **census** command tells us basic information about a file. With the **-k** option, **census** will tell us the number of barlines, the number of rests, the number of notes, the highest and lowest notes, and the longest and shortest notes for a **kern input. The following commands can be used to determine this information for (1) a bassoon part, (2) all woodwind parts:

```
    extract -i '*Ifagot' ives | census -k
    extract -i '*ICww' ives | census -k
```

With the **midi** and **perform** commands, **extract** allows the user to hear particular parts. For example, the following command extracts the bass and soprano voices, translates them to `**MIDI` data, and plays the output:

```
extract -i '*Ibass,*Isopran' lassus | midi | perform
```

We might extract a particular part (such as the trumpet part) and use the **trans** command to transpose it to another key:

```
extract -i '*Itromp' purcell | trans -d +1 -c +2
```

In addition, we might extract a particular instrument or group of instruments for notational display using the **ms** command. The following command will extract the string parts and create a postscript file for displaying or printing.

```
extract -i '*ICstr' brahms | ms > brahms.ps
```

The UNIX **lpr** command can be used to print a file or input stream. Suppose we want to transpose the piano accompaniment for a song by Hugo Wolf up an augmented second, and then print the transposed part:

```
extract -i '*IGacmp' wolf | trans -d +1 -c +3 | ms | lpr
```

Extracting Spines that Meander

As we saw in Chapter 5, spines can move around via various spine-path interpretations. Changes of spine position will cause havoc when extracting by fields (the **-f** option); **extract** will generate an error message and terminate. With the **-i** option, **extract** will follow the material throughout the input.

Consider the following input:

```
**mip   **dip  **dip  **blip
A       a      b      x
A       a      b      x
*       *^     *      *
A       a1     a2     b      x
A       a1     a2     b      x
A       a1     a2     b      x
*-      *-     *-     *-     *-
```

Suppose we want to extract the second spine (the first `**dip`) spine. Using the field option (**-f**) will generate an error message since this spine splits. Similarly, using the interpretation (**-i**) option will fail because the output will contain *all* of the `**dip` spines.

The **extract** command provides a third **-p** option that traces specific spine *paths*. Like the **-f** option, the **-p** option requires one or more numbers indicating the *beginning* field position for the spine. The command

```
extract -p 2 ...
```

will generate the following output:

```
**dip
a
a
*^
a1      a2
a1      a2
a1      a2
*-      *-
```

In *spine-path mode,* the **extract** command follows a given spine starting at the beginning of the file, and traces the course of that spine throughout the input stream. If spine-path changes are encountered in the input (such as spine exchanges, spine merges, or spine splits) the output adapts accordingly. If the "nth" spine is selected, the output consists of the nth spine and follows the path of that spine throughout the input until it is terminated or the end-of-file is encountered. What begins as the nth column, may end up as some other column (or columns) in the input.

There are complex circumstances where the **-p** option will not guarantee an output that conforms to the Humdrum syntax. When using the **-p** option it is prudent to check the output using the **humdrum** command in order to ensure that the output is valid. A full discussion of the **-p** option is given in the *Humdrum Reference Manual.*

Field-Trace Extracting

For circumstances where the input is very complex, **extract** provides a *field-trace mode* (**-t** option) that allows the user to select any combination of data tokens from the input stream. The field-trace option is rarely used when extracting spines. Refer to the *Humdrum Reference Manual* for further information.

Extracting Passages: The *yank* Command

A useful companion to the **extract** command is the Humdrum **yank** command. The **yank** command can be used to selectively extract segments or passages from a Humdrum input. The yanked material can be identified by absolute line numbers, or relative to some marker. In addition, **yank** is able to output logical segments, such as measures, phrases, or labelled sections, and is able to output material according to content. The output always consists of complete records; **yank** never outputs partial contents of a given input record.

The **yank** command provides five different ways of extracting material. The simplest way of yanking material is by specifying ranges of line numbers. In the following command, the **-l** option invokes the line-number operation. The **-r** option is used to specify the range. Ranges are defined by integers separated by commas, or with a dash indicating a range of consecutive values. For example, the following command selects lines, 5, 13, 23, 24, 25 and 26 from the file named `casel-la`:

```
    yank -l -r 5,13,23-26 casella
```

The dollar sign ($) can be used to refer to the last record in the input. For example, the following command yanks the first and last records from the file `mossolov`.

```
    yank -l -r '1,$' mossolov
```

Once again note that single quotes are needed here in order to prevent the shell from misinterpreting characters such as the dollar sign or the asterisk. Records close to the end of the input can be specified by subtracting some value from $. For example, the following command yanks the first 20 records from the last 30 records contained in the file `ginastera`. Notice that the dash/minus sign is used both to convey a range and as an arithmetic operator.

```
    yank -l -r '$-30-$-10' ginastera
```

If **yank** is given a Humdrum input, it always produces a syntactically correct Humdrum output. All interpretations prior to and within the yanked material are echoed in the output. The **yank** command also appends the appropriate spine-path terminators at the end of the yanked segment. By way of example, if we yanked line 10 (containing 4 spines) and line 100 (containing 5 spines), **yank** will include in the output the appropriate spine-path interpretations that specify how 4 spines became 5 spines.

Yanking by Marker

Alternatively, **yank** can output lines relative to some user-defined *marker.* This mode of operation can be invoked using the **-m** option. Markers are specified using regular expressions. The range option (**-r**) specifies which lines are to be output whenever a marker is encountered. For example, the following command outputs the first and third data records following each occurrence of the string "XXX" in the file `wieck`.

```
    yank -m XXX -r 1,3 wieck
```

If the value zero is specified in the range, the record containing the marker is itself output.

Since markers are interpreted by **yank** as regular expressions, complex markers can be defined. For example, the following command yanks the first data record following any record in the file `franck` beginning with a letter and ending with a number:

```
    yank -m '^[a-zA-Z].*[0-9]$' -r 1 franck
```

Using **yank -m** with a range defined as zero is an especially useful construction:

```
    yank -m regexp -r 0
```

This command is analogous to the familiar **grep** command. However, the output from **yank** will preserve all of the appropriate interpretations. In short, **yank** guarantees that the output conforms to the Humdrum syntax, whereas **grep** does not.

Suppose, for example, that we wanted to calculate the pitch intervals between notes that either begin or end a phrase in a monophonic input. If we use **grep** to search for `**kern` phrase indicators, we will be unable to process the resulting (non-Humdrum) output, since it will typically consist of just data records:

```
    grep [{}] sibelius
```

By contrast, the comparable **yank** command preserves the Humdrum syntax and so allows us to pipe the output to the melodic interval command:

```
    yank -m [{}] -r 0 sibelius | mint
```

Yanking by Delimiters

It is often convenient to yank material according to logical segments such as measures or phrases. In order to access such segments, the user must specify a segment *delimiter* using the **-o** option or the **-o** and **-e** options. For example, common system barlines are represented by the presence of an equals sign (=) at the beginning of a data token. Thus the user might yank specific measures from a file by defining the appropriate barline delimiter and providing a range of (measure) numbers. Consider the following command:

```
    yank -o ^= -r 1,12-13,25 joplin
```

This command will extract the first, twelfth, thirteenth and twenty-fifth measures from the file `joplin`. Unlike the **-m** option, the **-o** option interprets the range list as *ordinal* occurrences of segments delineated by the delimiter. Whole segments are output rather than specified records as is the case with **-m.** As in the case of markers, delimiters are interpreted according to regular expression syntax. Each input record containing the delimiter is regarded as the *start* of the next logical segment. In the above command, the range (**-r**) specifies that the first, twelfth, thirteenth, and twenty-fifth logical segments (measures) are to be yanked. All records starting with the delimiter record are output up to, but not including, the next occurrence of a delimiter record.

Where the input stream contains data prior to the first delimiter record, this data may be addressed as logical segment "zero." For example,

```
    yank -o ^= -r 0 mahler
```

can be used to yank all records prior to the first common system barline. Notice that *actual* measure numbers are irrelevant with the **-o** option: **yank** selects segments according to their *ordinal* position in the input stream rather than according to their *cardinal* label.

Not all segments are defined by a single marker. For example, unlike barlines, `**kern` phrases are marked by separate phrase-begin signifiers ('{') and phrase-end signifiers ('}'). The **-e** option for **yank** can be used to explicitly identify markers that *end* a segment. For example, the following command extracts the first four phrases in the file `tailleferre`:

```
    yank -o { -e } -r '1-4' tailleferre
```

When the **-n** option is invoked, however, **yank** expects a numerical value to be present in the input immediately following the user-specified delimiter. In this case, **yank** selects segments based on their numbered label rather than their ordinal position in the input. For example,

```
yank -n ^= -r 12 goldberg
```

will yank all segments beginning with the label `=12` in the input file `goldberg`. If more than one segment carries the specified segment number(s), all such segments are output. That is, if there are five measures labelled "measure 12", all five measures will be output. Note that the dollar sign anchor cannot be used in the range expression for the **-n** option. Note also that input tokens containing non-numeric characters appended to the number will have no effect on the pattern match. For example, input tokens such as `=12a`, `=12b`, or `=12;` will be treated as equivalent to `=12`.

As in the case of the **-o** option, a range of zero ('0') addresses material prior to the first delimiter record. (N.B. This behavior is unlike the **-m** option where zero addresses the record itself.) Like the **-o** option, the value zero may be reused for each specified input file. Thus, if `file1`, `file2` and `file3` are Humdrum files:

```
yank -n ^= -r 0 file1 file2 file3
```

will yank any leading (anacrusis) material in each of the three files.

Yanking by Section

When the **-s** option is invoked, **yank** extracts passages according to Humdrum section labels encoded in the input. Humdrum section labels will be described fully in Chapter 20. For now, we can simply note that section labels are tandem interpretations that conform to the syntax:

`*>`*label_name*

Label names can include any character except the tab. Labels are frequently used to indicate formal divisions, such as coda, exposition, bridge, second ending, trio, minuet, etc. The following command yanks the second instance of a section labelled `First Theme` in the file `mendelssohn`:

```
yank -s 'First Theme' -r 2 mendelssohn
```

Note that with "through-composed" Humdrum files it is possible to have more than one section containing the same section-label. Such situations are described in Chapter 20.

Examples Using *yank*

As mentioned earlier, **yank** will always produce a syntactically correct Humdrum output if given a proper Humdrum input. All interpretations prior to, and within, the yanked material are echoed in the output.

Any *comments* prior to the yanked passage may be included in the output by specifying the **-c** option.

The following examples illustrate how the **yank** command may be used.

```
    yank -l -r 1120 messiaen
```

yanks line 1120 in the file `messiaen`.

```
    yank -n ^= -r 27 sinfonia
```

yanks numbered measures 27 from the `**kern` file `sinfonia`.

```
    yank -n ^= -r 10-20 minuet waltz
```

yanks numbered measures 10 to 20 from both the `**kern` files `minuet` and `waltz`.

```
    yank -o ^= -r '0,$' fugue ricercar
```

yanks any initial anacrusis material plus the final measure of both `fugue` and `ricercar`.

```
    cat fugue ricercar | yank -o ^= -r '0,$'
```

yanks any initial anacrusis material from the file `fugue` followed by the final measure of `ricercar`.

```
    yank -n 'Rehearsal Marking ' -r 5-7 fugue ricercar
```

yanks segments beginning with the labels `"Rehearsal Marking 5"`, `"Rehearsal Marking 6"`, and `"Rehearsal Marking 7"`. Segments are deemed to end when a record is encountered containing the text `"Rehearsal Marking "`.

```
    yank -o { -e }  -r '1-$' webern
```

yanks all segments in the file `webern` beginning with a record containing "{" and ending with a record containing "}." The command:

```
    yank -o { -e } -r '1-4,$-3-$' faure
```

yanks the first four and last four segments in the file `faure`, where segments begin with an open brace ({) and end with a closed brace (}). In the `**kern` representation, this would extract the first four and last four phrases in the file.

```
    yank -s Coda -r 1 stamitz
```

will yank the first occurrence of a section labelled `Coda` in the file `stamitz`.

Note that yanked segments are output in exactly the order they appear in the input file. For example, assuming that measure numbers in an input stream increase sequentially, **yank** is unable to output measure number 6 prior to outputting measure number 5. The order of output material can be rearranged by invoked the **yank** command more than once (e.g. `yank -l -r 100 ...; yank -l -r 99 ...; yank -l -r 98 ...`).

Using *yank* in Pipelines

Like the other tools we have examined, **yank** can be profitably used in conjunction with other Humdrum tools. It is often useful to employ more than one **yank** in a pipeline. In the following command, the first **yank** isolates the 'Trio' section from the input file, and the second **yank** isolates the first four measures of the extracted Trio:

```
yank -s Trio dvorak | yank -o ^= 1-4
```

Similarly, we can link two **yank** commands to extract particular phrases from specified sections. For example, suppose we wanted to compare the first phrase of the exposition with the first phrase of the recapitulation:

```
yank -s Exposition haydn | yank -o { -e } -r 1 > Ephrase
yank -s Recapitulation haydn | yank -o { -e } -r 1 > Rphrase
```

Suppose we want to know how many notes there are in measures 8-16 in a `**kern` file named `borodin`.

```
yank -n = -r 8-16 borodin | census -k
```

Are there any subdominant chords between measures 80 and 86?

```
yank -n = -r 80-86 elgar | solfa | grep fa | grep la | grep do
```

How frequent is the dominant pitch in Strauss' horn parts?

```
extract -i '*Icor' strauss | solfa | grep -c so
```

Combining **yank** and **extract** can be especially useful. What is the highest note in the trumpet part in measure 29?

```
extract -i '*Itromp' tallis | yank -n = -r 29 | census -k
```

Also, we can combine **yank** with the **midi** and **perform** commands to hear particular sections. Play the Trio section in a Waldteufel waltz:

```
yank -s 'Trio' -r 1 waldteufel | midi | perform
```

Listen to the soprano clarinet part in the fourth and eighth phrases.

```
extract -i '*Iclars' quintet | yank -o { -e } -r 4,8 \
    | midi | perform
```

Note that when using **yank** to retrieve passages by markers (such as phrase marks), care must be taken since markers may be miscoordinated between several concurrent parts. Example 12.2 shows a passage that has overlapping phrases. When trying to extract a particular phrase for a particular part, the outputs will differ significantly depending on whether the **yank** command is invoked *before* or *after* the **extract** command.

Example 12.2. A Passage Containing Unsynchronized Phrases.

```
**kern   **kern
=1-      =1-
2r       8r
.        {8g
.        8a
.        8b
=2       =2
8r       4cc
{8e      .
8f       4dd}
8a       .
=3       =3
8g       {4ee
8e       .
4d}      4ff
=4       =4
*-       *-
```

The order of execution for some commands may cause some subtle differences. Suppose we wanted to identify the melodic intervals present in measures 8-32 for some work by Sibelius. The following two commands are likely to produce different results:

```
yank -n = -r 8-32 sibelius | mint
mint sibelius | yank -n = -r 8-32
```

In the second case, an interval will probably be calculated between between the last note of measure 7 and the first note of measure 8. This interval will be absent in the first case.

Reprise

In this chapter we have learned how to extract musical parts using **extract** and how to grab musical passages using **yank.** We saw that the **extract** command is also useful for isolating specific types of information, such as the lyrics, or ensuring that no other type of information is present in a data stream. In the case of **yank** we saw that passages can be extracted by defining arbitrary delimiters. In addition to extracting by measures, by sonorities, or by labelled sections, we can extract by rests, phrase marks — in fact, by any user-defined marker. We also saw how the command **yank -m** *regular-expression* **-r 0** can be used as a more sophisticated version of **grep** — a search tool that ensures the output will conform to the Humdrum syntax.

In the next chapter we will discuss how segments of music can be put back together again.