Humdrum Programming in C++

Craig Stuart Sapp

29 April/1 May 2009
Music 254
Stanford University

Downloading Code

- extra.humdrum.org/download
  A smaller set of programs with documentation.

- museinfo.sapp.org/doc/download
  A larger set of library code and examples

In linux:

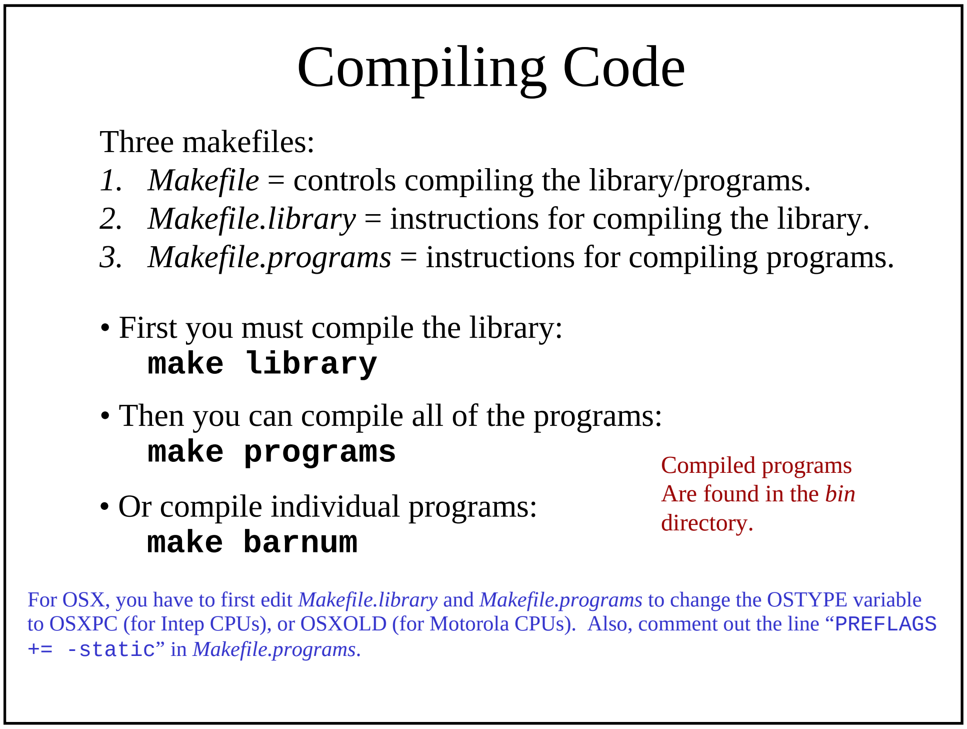
```
wget http://extra.humdrum.org/cgi-bin/humextra -O humextra-20090501.tar.bz2
tar xvjf humextra-20090501.tar.bz2
```

In Mac OSX:

```
curl http://extra.humdrum.org/cgi-bin/humextra -o humextra-20090501.tar.bz2
tar xvjf humextra-20090501.tar.bz2
```

Or if the above command doesn’t work:

```
bunzip2 humextra-20090501.tar.bz2; tar xvf humextra-20090501.tar
```



Compiling Code

Three makefiles:

1. *Makefile* = controls compiling the library/programs.
2. *Makefile.library* = instructions for compiling the library.
3. *Makefile.programs* = instructions for compiling programs.

- First you must compile the library:

```
make library
```

- Then you can compile all of the programs:

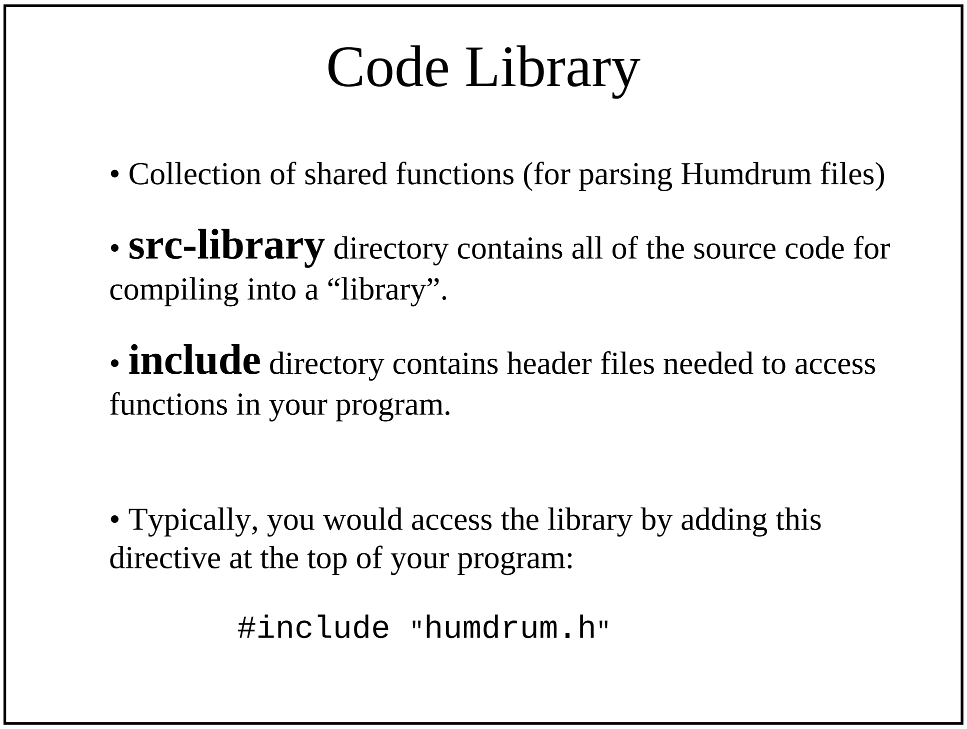
```
make programs
```

- Or compile individual programs:

```
make barnum
```

Compiled programs
Are found in the *bin*
directory.

For OSX, you have to first edit *Makefile.library* and *Makefile.programs* to change the OSTYPE variable to OSXPC (for Intep CPUs), or OSXOLD (for Motorola CPUs). Also, comment out the line “`PREFLAGS += -static`” in *Makefile.programs*.



Code Library

- Collection of shared functions (for parsing Humdrum files)

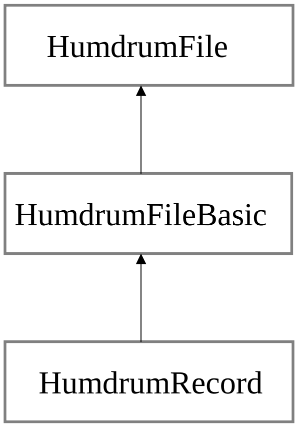
- **src-library** directory contains all of the source code for compiling into a “library”.

- **include** directory contains header files needed to access functions in your program.

- Typically, you would access the library by adding this directive at the top of your program:

```
#include "humdrum.h"
```

Main Library Classes



HumdrumFile

- Primary class for reading/writing and processing Humdrum data. Contains rhythm parsing functionality (**kern and **koto).

HumdrumFileBasic

- Basic Humdurm file parsing. Contains spine & null token analyses, but no knowledge of rhythm.

HumdrumRecord

- A HumdrumFile is basically an array of HumdrumRecords which are used to access data found on each line of the Humdrum file.

Other Useful Library Classes

Options

Class which can process command-line options and argument.

Convert

Class which can convert between different types of musical data, such as **kern pitch into MIDI or base-40.

Simplest Humdrum Program

```
#include "humdrum.h"
#include <iostream>
int main(int argc, char** argv) {
   HumdrumFile hfile;
   if (argc > 1) hfile.read(argv[1]);
   else hfile.read(std::cin);
   std::cout << hfile;
   return 0;
}
```

Read data from the first argument given on the command line (if there is one).

Otherwise, read from standard input if no filename is given to the program.

Print the contents of the file to standard output.

Save to a file called *humecho.cpp* and compile by typing “*make humecho*”. Program is stored in the *bin* directory. *Humecho.cpp* can be stored in the **humextra** directory or in the **humextra/src-programs** directory.

Running Your Program

- After compiling *humecho.cpp* successfully, type:

```
bin/humecho test.krn
```

- Where *test.krn* is a file with contents such as:

```
**kern
4e
4d
4c
4d
8e
8e
4e
*-
```

- Running bin/humecho should display the contents of the file on the screen.

- Also try the following command:

```
cat test.krn | bin/humecho
```

Accessing Individual Lines

humecho2.cpp

```
#include "humdrum.h"
int main(int argc, char** argv) {
   HumdrumFile hfile(argv[1]);
// HumdrumFile hfile;
// hfile.read(argv[1]);
   for (int i=0; i<hfile.getNumLines(); i++) {
      std::cout << hfile[i] << std::endl;
   }
   return 0;
}
```

```
i=0 **kern  **kern
  1 4r      4c
  2 8e      8g
  3 8f      8a
  4 4e      4r
  5 *-      *-
```

- HumdrumFile::**getNumLines**( )—returns count of lines in file.
- HumdrumFile::operator**[]**—accesses the nth line (HumdrumRecord).

Accessing Spine Data

humecho3.cpp

```
#include "humdrum.h"
int main(int argc, char** argv) {
   HumdrumFile hfile(argv[1]);
   for (int i=0; i<hfile.getNumLines(); i++) {
      std::cout << hfile[i][0];
      for (int j=1; j<hfile[i].getFieldCount(); j++) {
         std::cout << "\t" << hfile[i][j];
      }
      std::cout << std::endl;
   }
   return 0;
}
```

```
    j= 0     1
i=0 **kern  **kern
  1 4r      4c
  2 8e      8g
  3 8f      8a
  4 4e      4r
  5 *-      *-
```

hfile[i][j] is a const char*

HumdrumRecord::getFieldCount( ) returns spine count in line.

HumdrumRecord Line Types

see http://museinfo.sapp.org/examples/humdrum/hparser.cpp
and http://museinfo.sapp.org/include/Enum_humdrumRecord.h

```
hfile[i].getType()
```

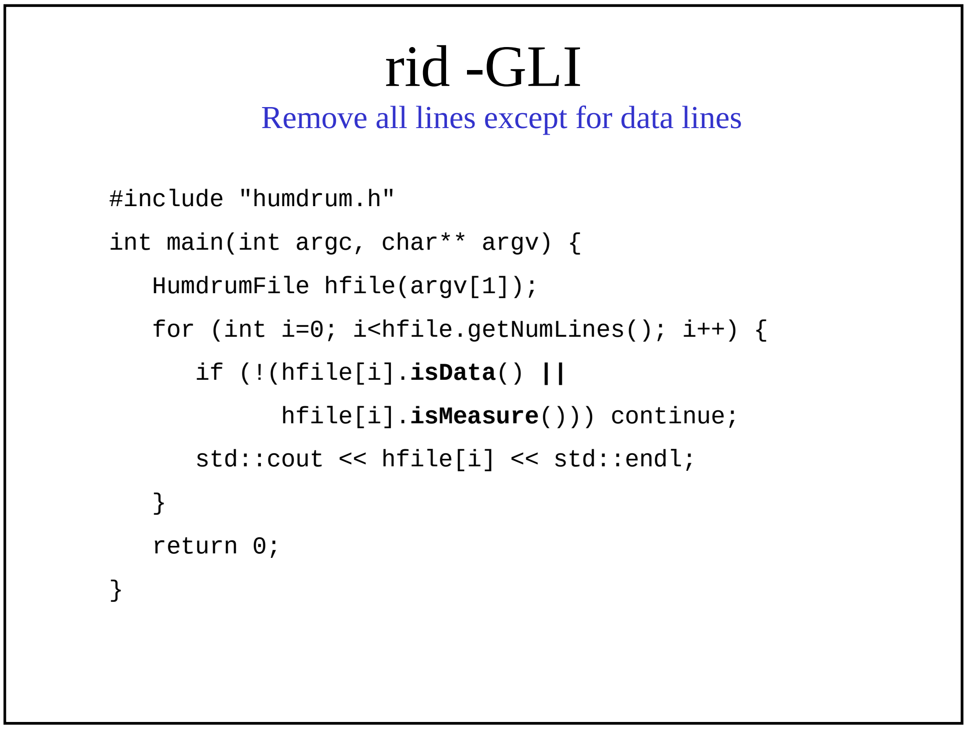
E_humrec_none	-- unknown line type
E_humrec_empty	-- empty line (technically invalid)
E_humrec_bibliography	-- of the form “!!!key: value”
E_humrec_global_comment	-- starts with “!!”
E_humrec_local_comment	-- local comment (!)
E_humrec_data_measure	-- line starting with “=”
E_humrec_interpretation	-- line starting with “*”
E_humrec_data	-- data lines other than measure

HumdrumRecord Line-Type Functions

.isData()	== true if data (other than barline).
.isMeasure()	== true if barline (line starts with “=”).
.isInterpretation()	== true if line starts with “*”.
.isBibliographic()	== true if in the form of “!!!key: value”.
.isGlobalComment()	== true if line starts with “!!” and not bib.
.isLocalComment()	== true if line starts with one “!”.
.isEmpty()	== true if nothing on line.

Composite tests:

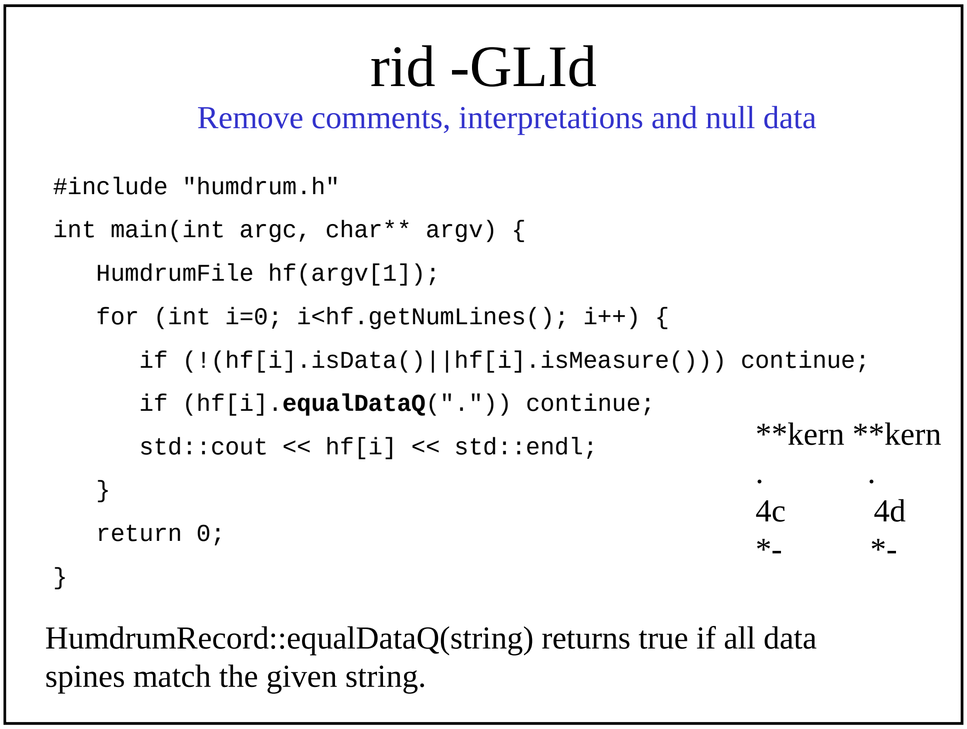
.isComment()	== isBibliographic || isGlobal… || isLocal…
.isTandem()	== Contains no Humdrum-file specific interpretations: *+, *-, *^, *v, *x, and exclusive interpretations.



rid -GLI

Remove all lines except for data lines

```
#include "humdrum.h"
int main(int argc, char** argv) {
   HumdrumFile hfile(argv[1]);
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!(hfile[i].isData() ||
            hfile[i].isMeasure())) continue;
      std::cout << hfile[i] << std::endl;
   }
   return 0;
}
```



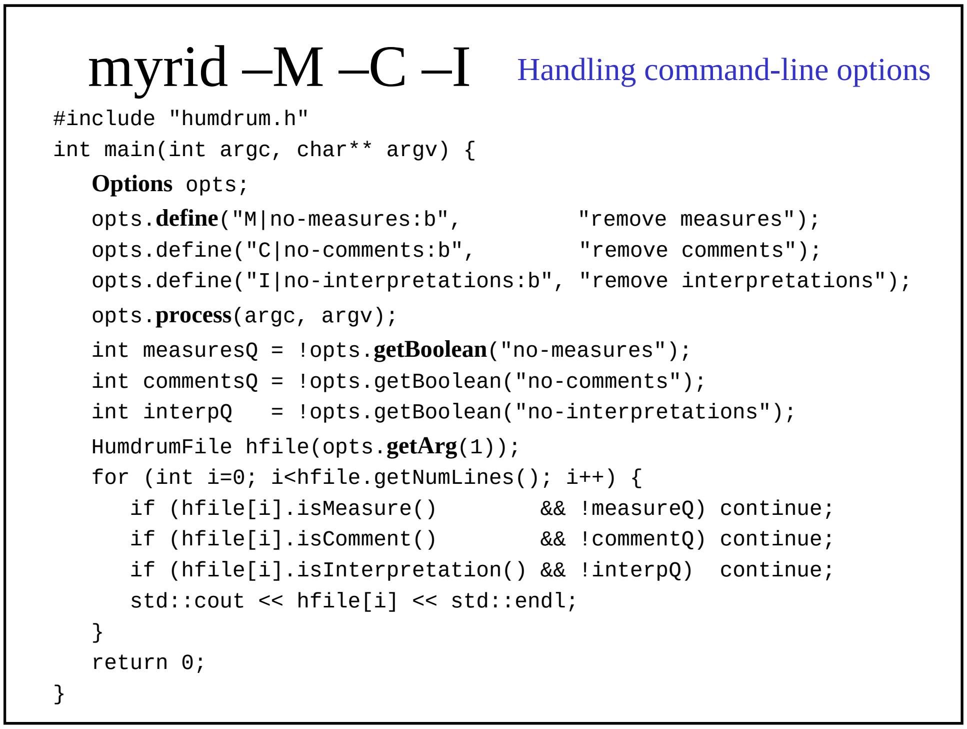
rid -GLId

Remove comments, interpretations and null data

```
#include "humdrum.h"
int main(int argc, char** argv) {
   HumdrumFile hf(argv[1]);
   for (int i=0; i<hf.getNumLines(); i++) {
      if (!(hf[i].isData()||hf[i].isMeasure())) continue;
      if (hf[i].equalDataQ(".")) continue;
      std::cout << hf[i] << std::endl;
   }
   return 0;
}
```

```
**kern **kern
.       .
4c      4d
*-      *-
```

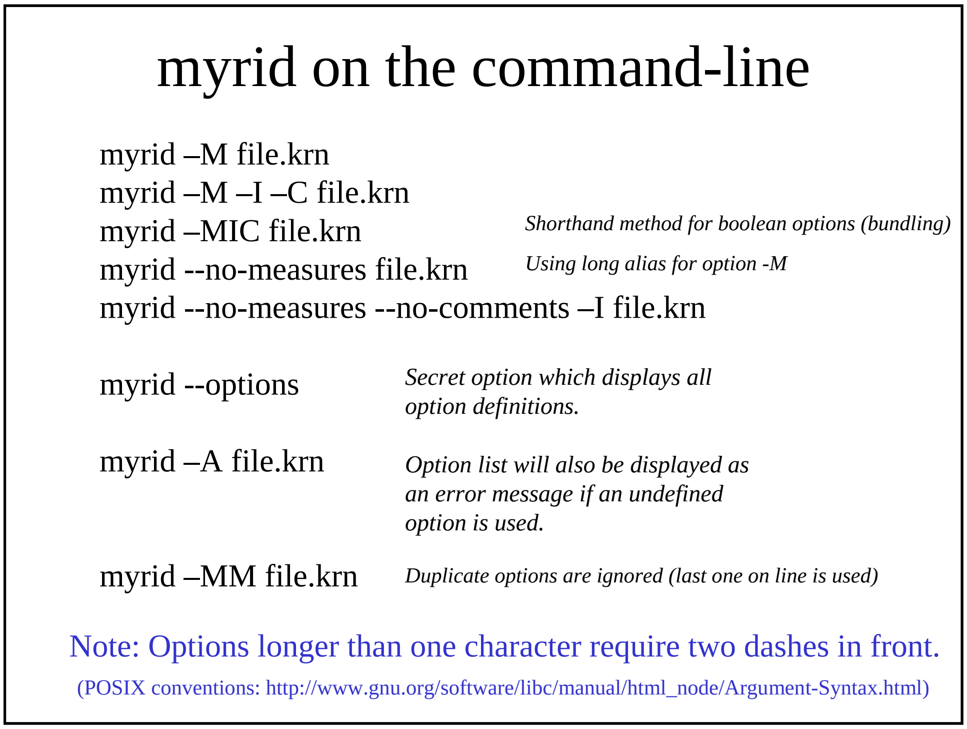
HumdrumRecord::equalDataQ(string) returns true if all data spines match the given string.



myrid –M –C –I

Handling command-line options

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options opts;
   opts.define("M|no-measures:b",         "remove measures");
   opts.define("C|no-comments:b",         "remove comments");
   opts.define("I|no-interpretations:b", "remove interpretations");
   opts.process(argc, argv);
   int measuresQ = !opts.getBoolean("no-measures");
   int commentsQ = !opts.getBoolean("no-comments");
   int interpQ   = !opts.getBoolean("no-interpretations");
   HumdrumFile hfile(opts.getArg(1));
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (hfile[i].isMeasure()        && !measureQ) continue;
      if (hfile[i].isComment()        && !commentQ) continue;
      if (hfile[i].isInterpretation() && !interpQ)  continue;
      std::cout << hfile[i] << std::endl;
   }
   return 0;
}
```



myrid on the command-line

myrid –M file.krn

myrid –M –I –C file.krn

myrid –MIC file.krn *Shorthand method for boolean options (bundling)*

myrid --no-measures file.krn *Using long alias for option -M*

myrid --no-measures --no-comments –I file.krn

myrid --options *Secret option which displays all option definitions.*

myrid –A file.krn *Option list will also be displayed as an error message if an undefined option is used.*

myrid –MM file.krn *Duplicate options are ignored (last one on line is used)*

Note: Options longer than one character require two dashes in front.

(POSIX conventions: http://www.gnu.org/software/libc/manual/html_node/Argument-Syntax.html)

More myrid on the command-line

Also legal syntax to place options after arguments when using Options class (non-POSIX):

```
myrid file1.krn –M
myrid –C file1.krn –M
myrid --no-interpretations file1.krn –MC
```

`opts.getArg(1)` will always return “file1.krn” in these cases.

- Suppose filename starts with a dash? (very bad to do, however):

```
myrid –M -- –file1.krn
```

Double dash forces end of options parsing, so you can’t add any options after filename in this case: myrid –M -- –file1.krn -C

Option Definitions

Options class designed for painless handling of command-line options.

`.define(`“*option definition string*”`,` “*brief option description*”`);`

Option definition string format:

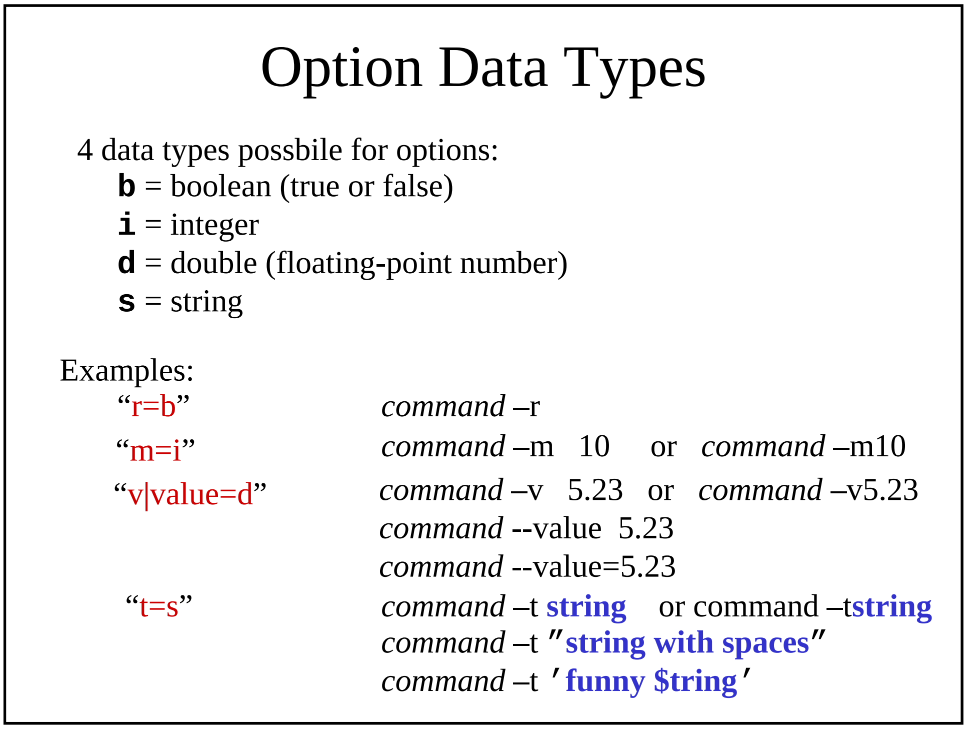
“Optionname = optiontype : defaultvalue”

Option name can contain *aliases*, which are separated by “|”. Examples:

`"M|no-measures=b"`

Option name: `M` *or* `no-measures`
Option Type: boolean
Default Value: booleans shouldn’t have default values

(technically they can, but you won’t be able to change them from the command-line)



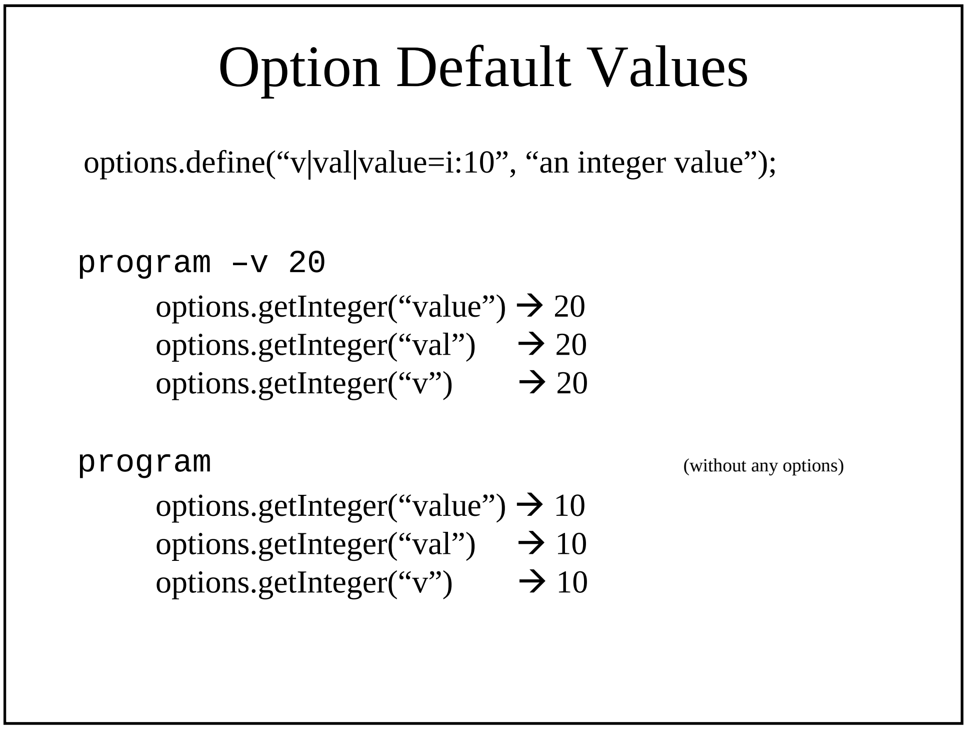
Option Data Types

4 data types possbile for options:

- **`b`** = boolean (true or false)
- **`i`** = integer
- **`d`** = double (floating-point number)
- **`s`** = string

Examples:

"r=b"	*command* –r
"m=i"	*command* –m 10 or *command* –m10
"v|value=d"	*command* –v 5.23 or *command* –v5.23 *command* --value 5.23 *command* --value=5.23
"t=s"	*command* –t **string** or command –t**string** *command* –t "**string with spaces**" *command* –t '**funny $tring**'



Option Default Values

options.define("v|val|value=i:10", "an integer value");

```
program –v 20
```

options.getInteger("value") → 20
options.getInteger("val") → 20
options.getInteger("v") → 20

```
program
```

(without any options)

options.getInteger("value") → 10
options.getInteger("val") → 10
options.getInteger("v") → 10

Extracting Option Values

.getBoolean(*option*)
.getInteger(*option*)
.getDouble(*option*)
.getString(*option*)

- All *get* functions can be applied to any type of option:

.define("t|temperature=d:80.6 farenheit", "temperature setting")

.getBoolean("temperature") → *true* if set via command-line
false if not set via command-line
(default value used in *false* case).
.getInteger ("temperature") → 80
.getDouble ("temperature") → 80.6
.getString ("temperature") → "80.6 farenheit"

Input from piped data or file(s)

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile;
   int numinputs = options.getArgCount();
   for (int i=1; i<=numinputs || i==0; i++) {
      if (numinputs < 1) {
         hfile.read(std::cin);  // read from standard input
      } else {
         hfile.read(options.getArg(i));
      }

      // do something with the Humdrum data here:
      std::cout << hfile;
   }
   return 0;
}
```

N.B.: Arguments indexed from 1 not 0. .getArg(0) returns the program name [same as .getCommand()]. .getArgCount() does not include the program name.

Command-line realizations:

humecho4 *file.krn*
humecho4 *file1.krn* *file2.krn*
cat *file.krn* | humecho4
humecho4

C String Comparisons

Like regular expressions (but no metacharacters)

#include <string.h>

strcmp(“*string1*”, “*string2*”)

returns 0 if strings are equivalent

return –1 if string1 is alphabetized before string2

return +1 if string1 is alphabetized after string2

strncmp(“*string1*”, “*string2*”, n)

compare only first *n* characters of strings.

strchr(“string”, ‘character’)

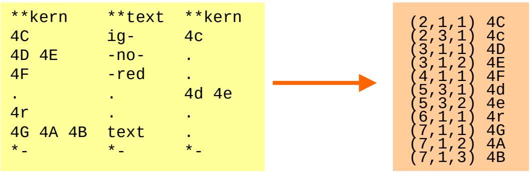
returns NULL (0) if character not found in string.

returns char* pointer to first character found.

Type “man strrchr” on the terminal for more information on strrchr.

Parsing Chords

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   char buffer[1024] = {0};
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;               // ignore non-data lines
      for (int j=0; j<hfile[i].getFieldCount(); j++) {
         if (strcmp("**kern", hfile[i].getExInterp(j)) != 0) continue;
         if (strcmp(".", hfile[i][j]) == 0) continue; // ignore null tokens
         int count = hfile[i].getTokenCount(j);
         for (int k=0; k<count; k++) {
            cout << "(" << i+1 <<"," << j+1 << "," << k+1 << ")\t"
                 << hfile[i].getToken(buffer, j, k) << endl;
         }
      }
   }
   return 0;
}
```



```
**kern    **text  **kern
4C        ig-     4c
4D 4E     -no-    .
4F        -red    .
.         .       4d 4e
4r        .       .
4G 4A 4B  text    .
*-        *-      *-
```

```
(2,1,1) 4C
(2,3,1) 4c
(3,1,1) 4D
(3,1,2) 4E
(4,1,1) 4F
(5,3,1) 4d
(5,3,2) 4e
(6,1,1) 4r
(7,1,1) 4G
(7,1,2) 4A
(7,1,3) 4B
```

Convert Class

- The convert class contains static functions for converting between different data types. View Convert.h for more info.

Example: Convert **kern note data into MIDI note numbers:

Convert::kernToMidiNoteNumber(“4d-”) → 61

Convert::base12ToKern(buffer, 61) → “c#”

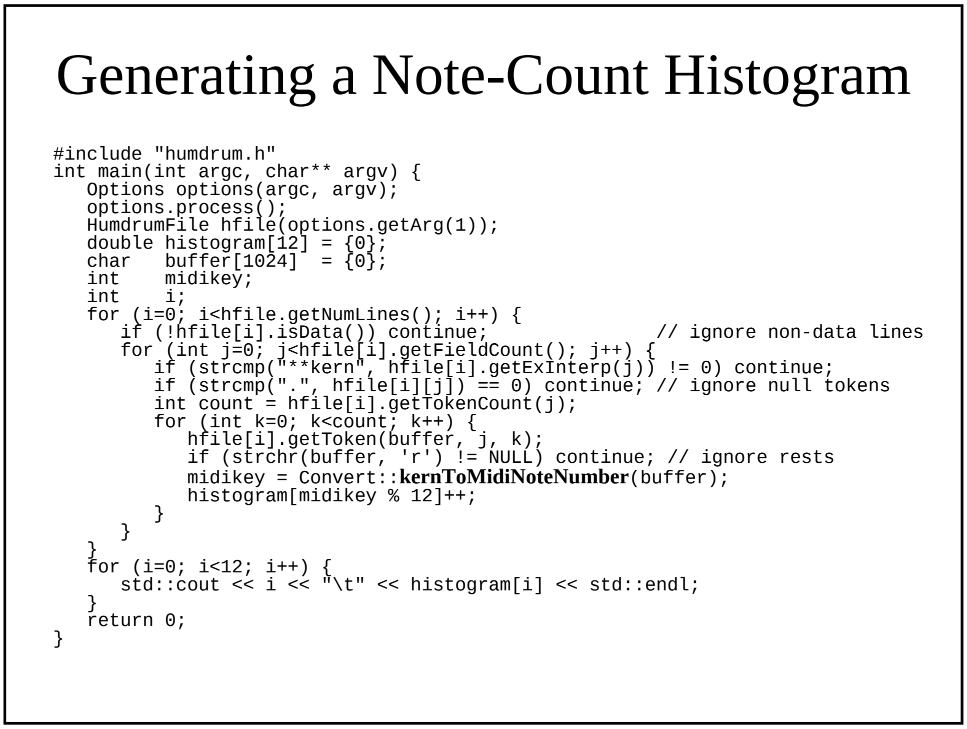
Example use of Convert

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;
      for (int j=0; j<hfile[i].getFieldCount(); j++) {
         if (strcmp("**kern", hfile[i].getExInterp(j)) != 0) continue;
         if (strcmp(".", hfile[i][j]) == 0) continue;     // ignore null tokens
         if (strchr(hfile[i][j], 'r') != NULL) continue;  // ignore rests
         cout << hfile[i][j] << "\t"
              << Convert::kernToMidiNoteNumber(hfile[i][j]) << endl;
      }
   }
   return 0;
}
```

```
**kern    **text  **kern
4C        ig-     4c
4D 4E     -no-    .
4F        -red    .
.         .       4d 4e
4r        .       .
4G 4A 4B  text    .
*-        *-      *-
```

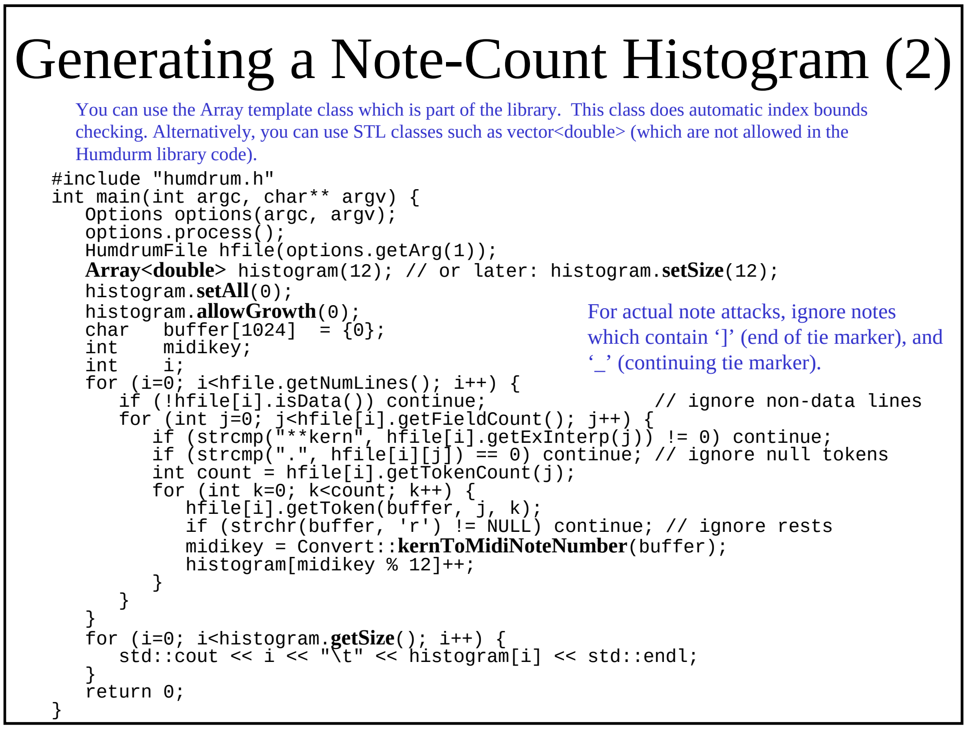
→

```
4C        48
4c        60
4D 4E     50
4F        53
4d 4e     62
4G 4A 4B  55
```



Generating a Note-Count Histogram

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   double histogram[12] = {0};
   char   buffer[1024]  = {0};
   int    midikey;
   int    i;
   for (i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;               // ignore non-data lines
      for (int j=0; j<hfile[i].getFieldCount(); j++) {
         if (strcmp("**kern", hfile[i].getExInterp(j)) != 0) continue;
         if (strcmp(".", hfile[i][j]) == 0) continue; // ignore null tokens
         int count = hfile[i].getTokenCount(j);
         for (int k=0; k<count; k++) {
            hfile[i].getToken(buffer, j, k);
            if (strchr(buffer, 'r') != NULL) continue; // ignore rests
            midikey = Convert::kernToMidiNoteNumber(buffer);
            histogram[midikey % 12]++;
         }
      }
   }
   for (i=0; i<12; i++) {
      std::cout << i << "\t" << histogram[i] << std::endl;
   }
   return 0;
}
```



Generating a Note-Count Histogram (2)

You can use the Array template class which is part of the library. This class does automatic index bounds checking. Alternatively, you can use STL classes such as vector<double> (which are not allowed in the Humdurm library code).

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   Array<double> histogram(12); // or later: histogram.setSize(12);
   histogram.setAll(0);
   histogram.allowGrowth(0);
   char   buffer[1024]  = {0};
   int    midikey;
   int    i;
   for (i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;               // ignore non-data lines
      for (int j=0; j<hfile[i].getFieldCount(); j++) {
         if (strcmp("**kern", hfile[i].getExInterp(j)) != 0) continue;
         if (strcmp(".", hfile[i][j]) == 0) continue; // ignore null tokens
         int count = hfile[i].getTokenCount(j);
         for (int k=0; k<count; k++) {
            hfile[i].getToken(buffer, j, k);
            if (strchr(buffer, 'r') != NULL) continue; // ignore rests
            midikey = Convert::kernToMidiNoteNumber(buffer);
            histogram[midikey % 12]++;
         }
      }
   }
   for (i=0; i<histogram.getSize(); i++) {
      std::cout << i << "\t" << histogram[i] << std::endl;
   }
   return 0;
}
```

For actual note attacks, ignore notes which contain ‘]’ (end of tie marker), and ‘_’ (continuing tie marker).

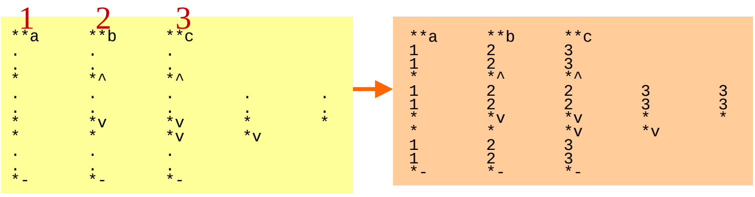
Duration-Weighted Note Histogram

Similar output to “key -f”

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   Array<double> histogram(12);
   histogram.setAll(0);
   histogram.allowGrowth(0);
   char   buffer[1024]  = {0};
   double duration;
   int    midikey;
   int    i;
   for (i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;             // ignore non-data lines
      for (int j=0; j<hfile[i].getFieldCount(); j++) {
         if (strcmp("**kern", hfile[i].getExInterp(j)) != 0) continue;
         if (strcmp(".", hfile[i][j]) == 0) continue; // ignore null tokens
         int count = hfile[i].getTokenCount(j);
         for (int k=0; k<count; k++) {
            hfile[i].getToken(buffer, j, k);
            if (strchr(buffer, 'r') != NULL) continue; // ignore rests
            midikey = Convert::kernToMidiNoteNumber(buffer);
            duration = Convert::kernToDuration(buffer);
            histogram[midikey % 12] += duration;
         }
      }
   }
   for (i=0; i<histogram.getSize(); i++) {
      std::cout << i << "\t" << histogram[i] << std::endl;
   }
   return 0;
}
```

Primary Spine Enumeration

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) {
         std::cout << hfile[i] << std::endl;
         continue;
      }
      std::cout << hfile[i].getPrimaryTrack(0);
      for (int j=1; j<hfile[i].getFieldCount(); j++) {
         std::cout << '\t' << hfile[i].getPrimaryTrack(j);
      }
      std::cout << endl;
   }
   return 0;
}
```



```
**a     **b     **c
.       .       .
.       .       .
*       *^      *^
.       .       .       .       .
.       .       .       .       .
*       *v      *v      *       *
*       *       *v      *v
.       .       .
.       .       .
*-      *-      *-
```

```
**a     **b     **c
1       2       3
1       2       3
*       *^      *^
1       2       2       3       3
1       2       2       3       3
*       *v      *v      *       *
*       *       *v      *v
1       2       3
1       2       3
*-      *-      *-
```

hfile.getMaxTrack() → 3

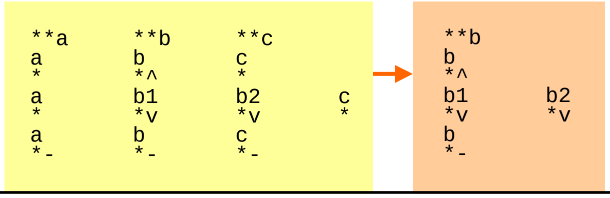
myextract.cpp (1): extract

```
#include "humdrum.h"

void extract(HumdrumFile& hfile, int primarytrack) {
   int i, j, fcount, pcount;
   for (i=0; i<hfile.getNumLines(); i++) {
      switch (hfile[i].getType()) {
         case E_humrec_local_comment:   case E_humrec_data_measure:
         case E_humrec_interpretation:  case E_humrec_data:
            fcount= hfile[i].getFieldCount();
            pcount = 0;
            for (j=0; j<fcount; j++) {
               if (primarytrack == hfile[i].getPrimaryTrack(j)) {
                  if (pcount++ > 0) cout << '\t';
                  cout << hfile[i][j];
               }
            }
            if (pcount > 0) cout << endl;
            break;
         default:
            cout << hfile[i] << endl;
      }
   }
}
```

myextract.cpp (2): main

```
int main(int argc, char** argv) {
   Options opts;
   opts.define("f|field=i:0", "extract specified spine");
   opts.process(argc, argv);
   int primarytrack = opts.getInteger("field");
   int numinputs = opts.getArgCount();
   HumdrumFile hfile;
   for (int i=1; i<=numinputs || i==0; i++) {
      if (numinputs < 1) {
         hfile.read(std::cin);  // read from standard input
      } else {
         hfile.read(opts.getArg(i));
      }
      extract(hfile, primarytrack);
   }
   return 0;
}
```

bin/myextract –f 2 file.krn



```
**a      **b      **c
a        b        c
*        *^       *
a        b1       b2       c
*        *v       *v       *
a        b        c
*-       *-       *-
```

```
**b
b
*^
b1       b2
*v       *v
b
*-
```

Spine Manipulation History

```
#include "humdrum.h"
int main(int argc, char** argv) {
   Options options(argc, argv);
   options.process();
   HumdrumFile hfile(options.getArg(1));
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) {
         std::cout << hfile[i] << std::endl;
         continue;
      }
      std::cout << hfile[i].getSpineInfo(0);
      for (int j=1; j<hfile[i].getFieldCount(); j++) {
         std::cout << "\t" << hfile[i].getSpineInfo(j);
      }
      std::cout << endl;
   }
   return 0;
}
```

```
**a  **b  **c
.    .    .
*    *^   *^
.    .    .    .    .
*    *    *^   *    *
.    .    .    .    .    .
*    *    *v   *v   *    *
*    *v   *v   *    *
.    .    .    .
*    *    *v   *v
.    .    .
*-   *-   *-
```

```
**a **b  **c
1   2    3
*   *^   *^
1   (2)a (2)b     (3)a    (3)b
*   *    *^       *       *
1   (2)a ((2)b)a ((2)b)b (3)a (3)b
*   *    *v       *v      *    *
*   *v   *v       *       *
1   2    (3)a     (3)b
*   *    *v       *v
1   2    3
*-  *-   *-
```

Spine Manipulation (2)

Split / Join

*^ / *v

```
**a     **b     **c
.       .       .
*       *^      *
.       .       .       .
*       *       *^      *
.       .       .       .       .
*       *v      *v      *v      *
.       .       .
*-      *-      *-
```

```
**a     **b     **c
1       2       3
*       *^      *
1       (2)a    (2)b    3
*       *       *^      *
1       (2)a    ((2)b)a ((2)b)b 3
*       *v      *v      *v      *
1       2       3
*-      *-      *-
```

add / end

*+ / *-

```
**a     **b
.       .
*       *+      **c
.       .       .
*       *-      *
.       .
*-      *-
```

```
**a     **b
1       2
*       *+      **c
1       2       3
*       *-      *
1       3
*-      *-
```

exchange

*x

```
**a     **b
.       .
.       .
*x       *x
.       .
.       .
*-      *-
```

```
**a     **b
1       2
1       2
*x      *x
2       1
2       1
*-      *-
```

Spine Manipulation (3)

```
**a **b
.   .
*   *^
.   .   .
*+  **c *
.   .   .   .
*   *   *x  *x
.   .   .   .
*   *   *^  *
.   .   .   .   .
*   *+  **d *   *
.   .   .   .   .   .
*   *   *   *   *x  *x
.   .   .   .   .   .
*   *-  *   *   *   *
.   .   .   .   .
*v  *v  *   *   *
.   .   .   .
*   *v  *v  *
.   .   .
*   *v  *v
.   .
*-  *-
```

```
**a  **b
1    2
*    *^
1    (2)a         (2)b
*+   **c          *
1    3            (2)a     (2)b
*    *            *x       *x
1    3            (2)b     (2)a
*    *            *^       *
1    3            ((2)b)a  ((2)b)b  (2)a
*    *+           **d      *        *
1    3            4        ((2)b)a  ((2)b)b  (2)a
*    *            *        *        *x       *x
1    3            4        ((2)b)a  (2)a     ((2)b)b
*    *-           *        *        *        *
1    4            ((2)b)a  (2)a     ((2)b)b
*v   *v           *        *        *
1 4  ((2)b)a      (2)a     ((2)b)b
*    *v           *v       *
1 4  ((2)b)a (2)a ((2)b)b
*    *v           *v
1 4  2
*-   *-
```

Regular Expressions in C

(GNU implementation of POSIX specification used. Should work on any linux computer. But be careful, there are other non-compatible implementations.)

```
#include <regex.h>
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
   if (argc < 3) exit(1);
   const char *searchstring = argv[1];
   const char *datastring   = argv[2];
   regex_t re;
   int flags = 0 | REG_EXTENDED | REG_ICASE;
   int status = regcomp(&re, searchstring, flags);
   if (status !=0) {
      char errstring[999];
      regerror(status, &re, errstring, 999);
      cerr << errstring << endl;
      exit(1);
   }
   status = regexec(&re, datastring, 0, NULL, 0);
   if (status == 0) cout << "Match Found" << endl;
   else             cout << "Match Not Found" << endl;
}
```

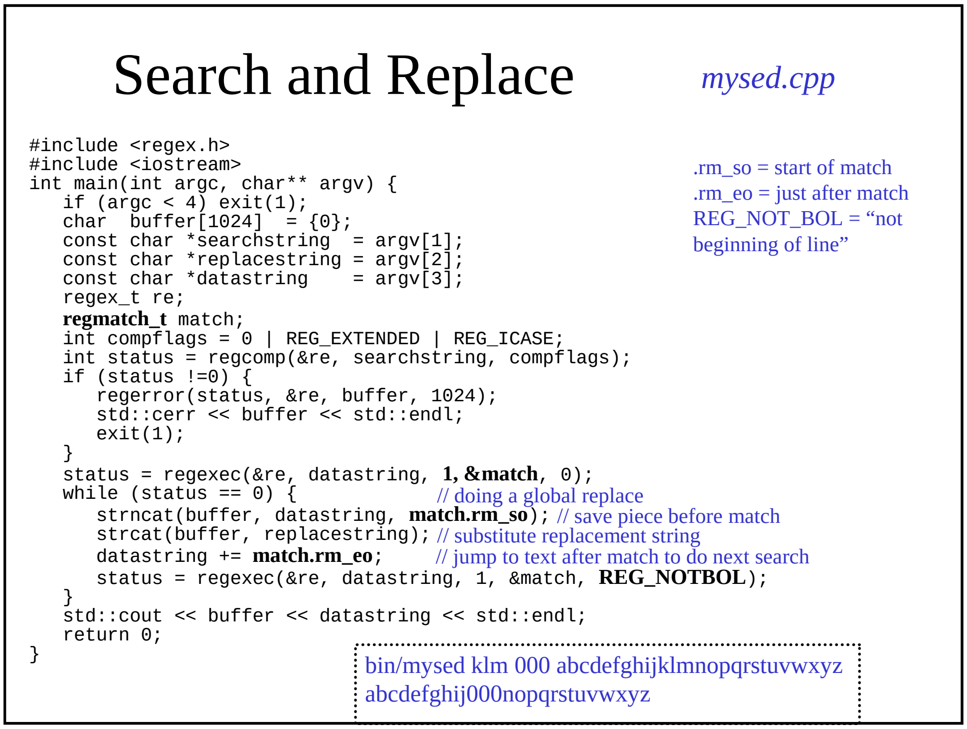
→ First thing on command line is the search string

→ second thing is data string to search

REG_EXTENDED: Use extended regular expression syntax

REG_ICASE: Ignore capitalization (upper- and lowercase text will match)

Regexec returns: 0 if matched; status != 0 if didn’t match.



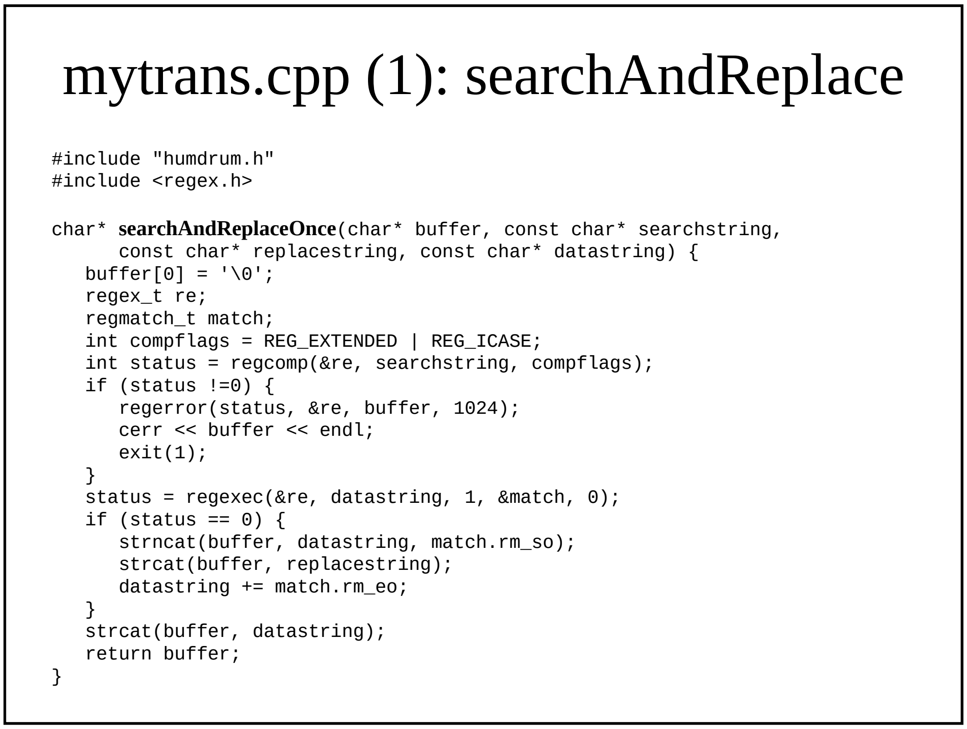
Search and Replace

mysed.cpp

.rm_so = start of match
.rm_eo = just after match
REG_NOT_BOL = “not beginning of line”

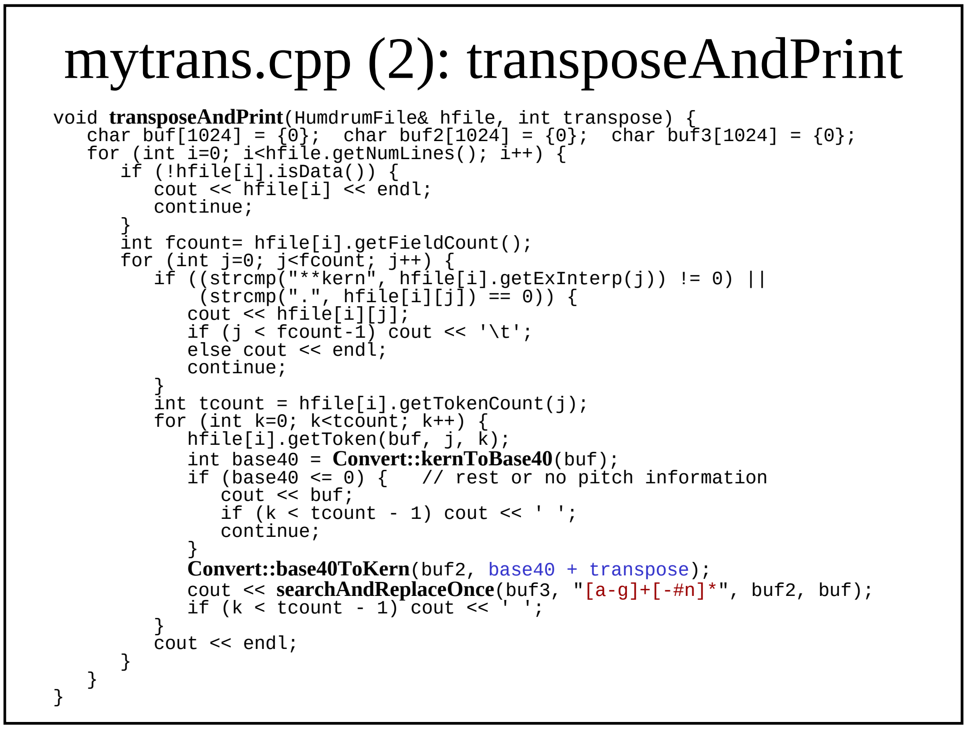
```
#include <regex.h>
#include <iostream>
int main(int argc, char** argv) {
   if (argc < 4) exit(1);
   char  buffer[1024]  = {0};
   const char *searchstring  = argv[1];
   const char *replacestring = argv[2];
   const char *datastring    = argv[3];
   regex_t re;
   regmatch_t match;
   int compflags = 0 | REG_EXTENDED | REG_ICASE;
   int status = regcomp(&re, searchstring, compflags);
   if (status !=0) {
      regerror(status, &re, buffer, 1024);
      std::cerr << buffer << std::endl;
      exit(1);
   }
   status = regexec(&re, datastring, 1, &match, 0);
   while (status == 0) {                    // doing a global replace
      strncat(buffer, datastring, match.rm_so); // save piece before match
      strcat(buffer, replacestring); // substitute replacement string
      datastring += match.rm_eo;      // jump to text after match to do next search
      status = regexec(&re, datastring, 1, &match, REG_NOTBOL);
   }
   std::cout << buffer << datastring << std::endl;
   return 0;
}
```

```
bin/mysed klm 000 abcdefghijklmnopqrstuvwxyz
abcdefghij000nopqrstuvwxyz
```



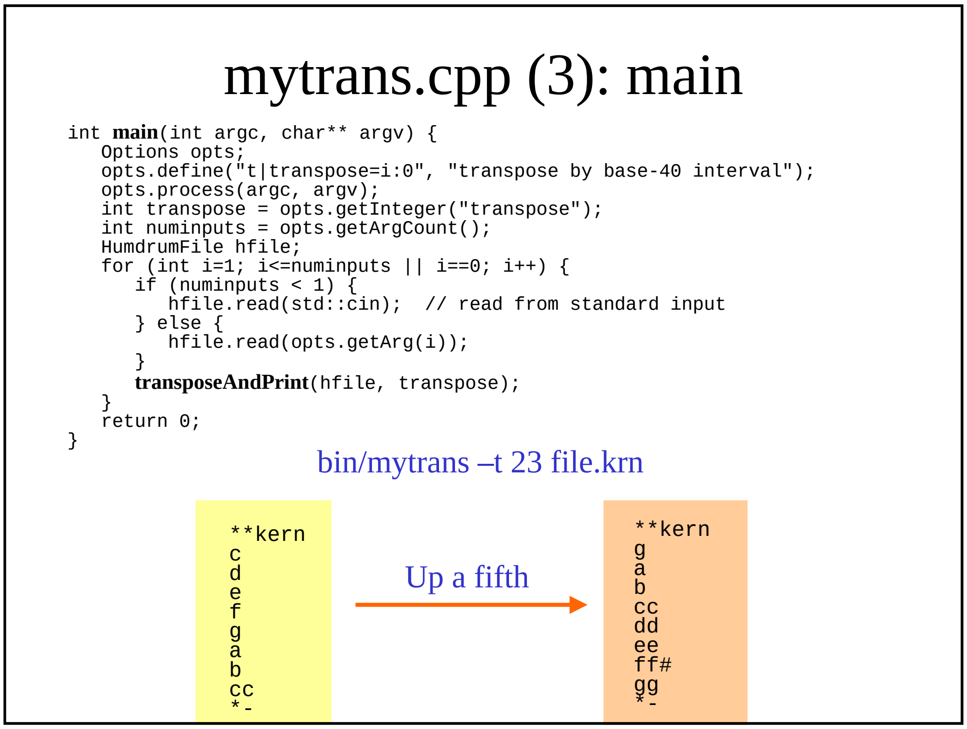
mytrans.cpp (1): searchAndReplace

```
#include "humdrum.h"
#include <regex.h>

char* searchAndReplaceOnce(char* buffer, const char* searchstring,
      const char* replacestring, const char* datastring) {
   buffer[0] = '\0';
   regex_t re;
   regmatch_t match;
   int compflags = REG_EXTENDED | REG_ICASE;
   int status = regcomp(&re, searchstring, compflags);
   if (status !=0) {
      regerror(status, &re, buffer, 1024);
      cerr << buffer << endl;
      exit(1);
   }
   status = regexec(&re, datastring, 1, &match, 0);
   if (status == 0) {
      strncat(buffer, datastring, match.rm_so);
      strcat(buffer, replacestring);
      datastring += match.rm_eo;
   }
   strcat(buffer, datastring);
   return buffer;
}
```



mytrans.cpp (2): transposeAndPrint

```
void transposeAndPrint(HumdrumFile& hfile, int transpose) {
   char buf[1024] = {0};  char buf2[1024] = {0};  char buf3[1024] = {0};
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) {
         cout << hfile[i] << endl;
         continue;
      }
      int fcount= hfile[i].getFieldCount();
      for (int j=0; j<fcount; j++) {
         if ((strcmp("**kern", hfile[i].getExInterp(j)) != 0) ||
             (strcmp(".", hfile[i][j]) == 0)) {
            cout << hfile[i][j];
            if (j < fcount-1) cout << '\t';
            else cout << endl;
            continue;
         }
         int tcount = hfile[i].getTokenCount(j);
         for (int k=0; k<tcount; k++) {
            hfile[i].getToken(buf, j, k);
            int base40 = Convert::kernToBase40(buf);
            if (base40 <= 0) {   // rest or no pitch information
               cout << buf;
               if (k < tcount - 1) cout << ' ';
               continue;
            }
            Convert::base40ToKern(buf2, base40 + transpose);
            cout << searchAndReplaceOnce(buf3, "[a-g]+[-#n]*", buf2, buf);
            if (k < tcount - 1) cout << ' ';
         }
         cout << endl;
      }
   }
}
```



mytrans.cpp (3): main

```
int main(int argc, char** argv) {
   Options opts;
   opts.define("t|transpose=i:0", "transpose by base-40 interval");
   opts.process(argc, argv);
   int transpose = opts.getInteger("transpose");
   int numinputs = opts.getArgCount();
   HumdrumFile hfile;
   for (int i=1; i<=numinputs || i==0; i++) {
      if (numinputs < 1) {
         hfile.read(std::cin);  // read from standard input
      } else {
         hfile.read(opts.getArg(i));
      }
      transposeAndPrint(hfile, transpose);
   }
   return 0;
}
```

bin/mytrans –t 23 file.krn

```
**kern
c
d
e
f
g
a
b
cc
*-
```

Up a fifth

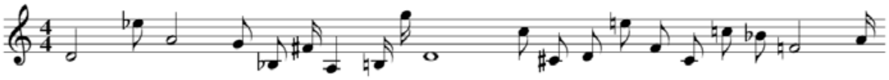
```
**kern
g
a
b
cc
dd
ee
ff#
gg
*-
```

Random Melody

drand48() = random number in range from 0.0 to 1.0.
srand() = seed random number generator.

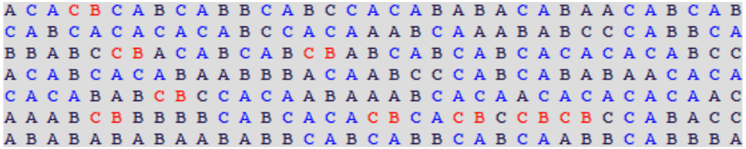
```
#include "humdrum.h"
#include <stdlib.h>     /* for drand48 random numbers */
#include <time.h>       /* for time(NULL) function */

void printRandomMelody(int notecount, int seed) {
   cout << "!!!seed:\t" << seed << endl;
   cout << "**kern\n";
   int pitch, rhythm;
   char buffer[1024] = {0};
   for (int i=0; i<notecount; i++) {
      rhythm = int(drand48() * 16 + 1 + 0.5);
      pitch  = int(drand48() * 24 + 12*4.5 + 3);
      cout << rhythm << Convert::base12ToKern(buffer, pitch) << endl;
   }
   cout << "*-\n";
}
int main(int argc, char** argv) {
   Options options;
   options.define("c|count=i:20", "number of notes to generate");
   options.define("s|seed=i:-1",  "random number generator seed");
   options.process(argc, argv);
   int seed = options.getInteger("seed");
   if (seed < 0) {
      seed = time(NULL);  // time in seconds since 1 Jan 1970
   }
   srand48(seed);
   printRandomMelody(options.getInteger("count"), seed);
   return 0;
}
```

Random Melody (2)

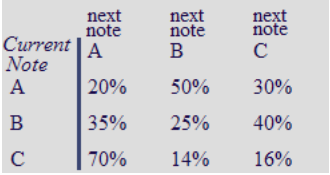


```
!!!seed: 1241137496
**kern
2d
9ee-
2a
13g
15B-
16f#
7A
16B
17gg
1d
11cc
14c#
10d
11ee
12f#
13c#
8cc
14b-
3f
17a
*-
```

Markov Chains



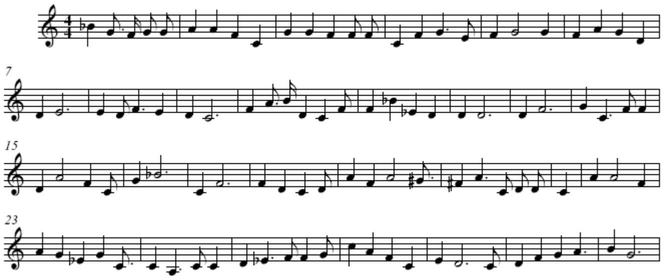
A C A C B C A B C A B B C A B C C A C A B A B A C A B A A C A B C A B
C A B C A C A C A C A B C C A C A A A B C A A A B A B C C C A B B C A
B B A B C C B A C A B C A B C B A B C A B C A B C A C A C A C A B C C
A C A B C A C A B A A B B B A C A A B C C C A B C A B A B A A C A C A
C A C A B A B C B C C A C A A B A A A B C A C A A C A C A C A C A A C
A A A B C B B B B B C A B C A C A C B C A C B C C B C B C C A B A C C
A B A B A B A B A A B A B B C A B C A B B C A B C A A B B C A B B B A



	next note	next note	next note
Current Note	A	B	C
A	20%	50%	30%
B	35%	25%	40%
C	70%	14%	16%

Markov Melody

bin/markovmelody `grep –l ’M4\/4’ ~/scores/nova/*.krn` -g 100



First-order markov analysis of input data: pitch class and metrical position, both done independently.

Markov Melody (2): buildTable

```
#include "humdrum.h"
#include <regex.h>
#include <stdlib.h>
#include <time.h>

void buildTable(HumdrumFile& hfile, Array<Array<double> >& ptable,
      Array<Array<double> >& mtable) {
   int lastmeter = -1; int lastpitch = -1;
   int meter, pitch;
   hfile.analyzeRhythm();
   for (int i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;
      if (strcmp("**kern", hfile[i].getExInterp(0)) != 0) continue;
      if (strcmp(hfile[i][0], ".") == 0) continue;    // ignore null tokens
      if (strchr(hfile[i][0], 'r') != NULL) continue; // ignore rests
      pitch = Convert::kernToBase40(hfile[i][0]) % 40;
      meter = int((hfile[i].getBeat() - 1.0) * 4 + 0.5);
      if (meter < 0) meter = 0;
      if (meter >= 40) meter = 39;
      if (lastmeter < 0) {
         lastpitch = pitch; lastmeter = meter;
         continue;
      }
      mtable[lastmeter][meter]++;   mtable[lastmeter][40]++;
      ptable[lastpitch][pitch]++;   ptable[lastpitch][40]++;
      lastpitch = pitch;            lastmeter = meter;
   }
}
```

Markov Melody (3): printTables

```
void printTables(Array<Array<double> >& ptable,
   Array<Array<double> >& mtable, int style) {
   int i, j;
   double value;
   char buffer[32] = {0};
   for (i=0; i<ptable.getSize(); i++) {
      cout << '\t' << Convert::base40ToKern(buffer, i+4*40);
   }
   cout << endl;
   for (i=0; i<ptable.getSize(); i++) {
      cout << Convert::base40ToKern(buffer, i+4*40);
      for (j=0; j<40; j++) {
         value = style ? ptable[i][j]/ptable[i][40] : ptable[i][j];
         cout << '\t' << value;
      }
      cout << '\t' << ptable[i][40] << endl;
   }
   cout << endl;
   for (i=0; i<mtable.getSize(); i++) cout << "\tb" << i/4.0 + 1.0;
   cout << endl;
   for (i=0; i<mtable.getSize(); i++) {
      cout << "b" << i/4.0 + 1.0;
      for (j=0; j<mtable[i].getSize(); j++) cout << '\t' << mtable[i][j];
      cout << endl;
   }
}
```

Markov Melody (4): two functions

```
int chooseNextTransition(Array<Array<double> >& table, int state) {
   double target = drand48() * table[state][40];
   double sum = 0.0;
   for (int i=0; i<40; i++) {
      sum += table[state][i];
      if (sum > target)  return i;
   }
   return 39;
}

void smoothMelody(Array<double>& meldur, Array<int>& melpitch) {
   int beforei, afteri, inta, intb;
   for (int i=2; i<meldur.getSize()-2; i++) {
      if (meldur[i] < 0.0) continue;
      afteri  = i+1;   beforei = i-1;
      if (meldur[afteri] < 0.0) afteri++;
      if (meldur[beforei] < 0.0) beforei--;
      inta = melpitch[i] - melpitch[beforei];
      intb = melpitch[i] - melpitch[afteri];
      if ((inta > 22) && (intb > 22)) {
         melpitch[i] -= 40;
      } else if ((inta < -22) && (intb < -22)) {
         melpitch[i] += 40;
      }
   }
}
```

Markov Melody (5): generateMelody

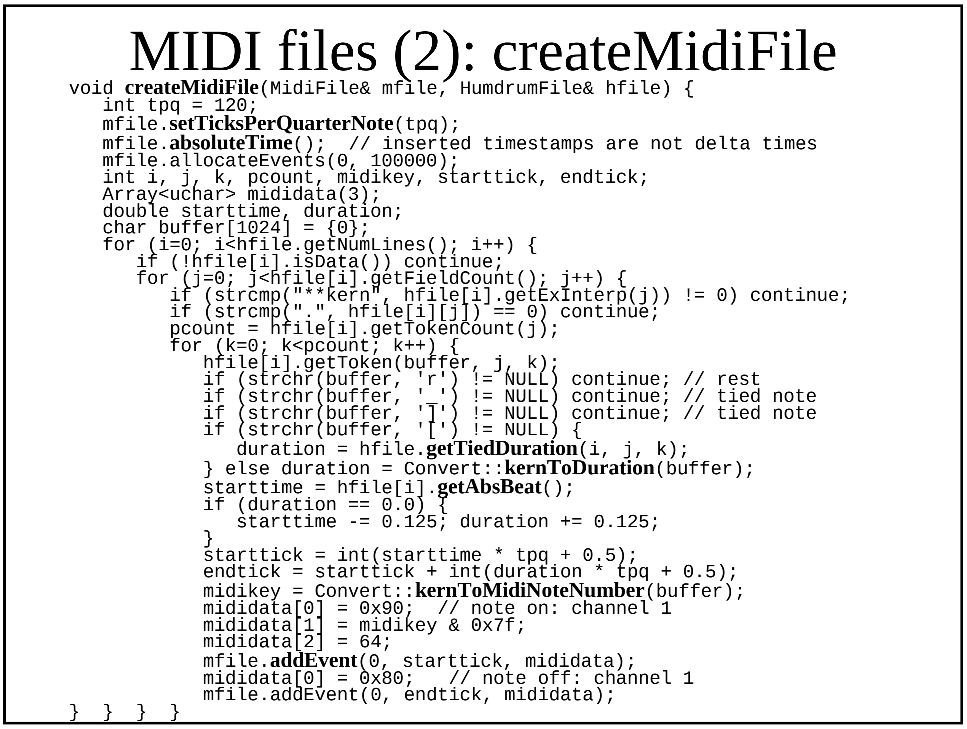
```
void generateMelody(Array<Array<double> >& ptable,
      Array<Array<double> >& mtable, int count) {
   int    pitch, pitchclass = 2, meter = 0, oldmeter = 0;
   int    i, measurenumber = 2;
   double duration, barmarker = -1;
   char   buffer[1024] = {0};
   Array<int> melpitch(count*2);  melpitch.setSize(0);
   Array<double> meldur(count*2); meldur.setSize(0);
   for (i=0; i<count; i++) {
      pitchclass  = chooseNextTransition(ptable, pitchclass);
      meter = chooseNextTransition(mtable, meter);
      if (meter > oldmeter) duration = (meter - oldmeter) / 4.0;
      else {
         duration = (4 + meter - oldmeter) / 4.0;
         meldur.append(barmarker);
         pitch = measurenumber++;
         melpitch.append(pitch);
      }
      oldmeter = meter;
      if (duration == 0.0) duration = 4.0;
      if (duration > 4.0)  duration = 4.0;
      if (duration < 0.0)  duration = 1.0;
      pitch = pitchclass + 4 * 40;
      meldur.append(duration);  melpitch.append(pitch);
   }
   smoothMelody(meldur, melpitch);
   cout << "**kern\n*M4/4\n=1-\n";
   for (i=0; i<meldur.getSize(); i++) {
      if (meldur[i] < 0.0) cout << "=" << melpitch[i] << endl;
      else {
         cout << Convert::durationToKernRhythm(buffer, meldur[i]);
         cout << Convert::base40ToKern(buffer, melpitch[i]);
         cout << endl;
      }
   }
   cout << "*-" << endl;
}
```

Markov Melody (6): main

```
int main(int argc, char** argv) {
   Options options;
   options.define("t|table=b",       "display table of transitions");
   options.define("f|fraction=b",    "display transitions as fractions");
   options.define("g|generate=i:20", "generate specified number of notes");
   options.process(argc, argv);
   srand48(time(NULL)); HumdrumFile hfile;
   Array<Array<double> > ptable; // pitch transition table
                               // (scale degrees would be musically better)
   Array<Array<double> > mtable; // meter transition table
   ptable.setSize(40); ptable.allowGrowth(0);
   mtable.setSize(40); mtable.allowGrowth(0);
   int i;
   for (i=0; i<ptable.getSize(); i++) {
      ptable[i].setSize(41); ptable[i].allowGrowth(0); ptable[i].setAll(0.0);
      mtable[i].setSize(41); mtable[i].allowGrowth(0); mtable[i].setAll(0.0);
   }
   int numinputs = options.getArgCount();
   for (i=1; i<=numinputs || i==0; i++) {
      if (numinputs < 1) hfile.read(std::cin);
      else hfile.read(options.getArg(i));
      buildTable(hfile, ptable, mtable);
   }
   if (options.getBoolean("table")) {
      printTables(ptable, mtable, options.getBoolean("fraction"));
   } else {
      generateMelody(ptable, mtable, options.getInteger("generate"));
   }
   return 0;
}
```

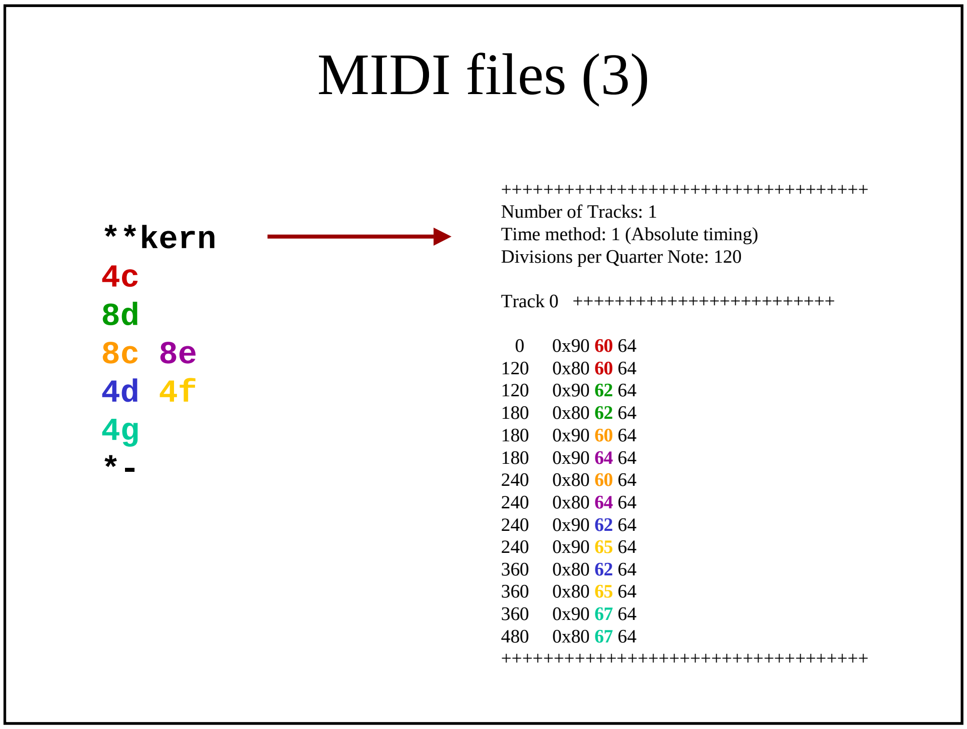
MIDI files *mysmf.cpp*

```
#include "MidiFile.h"
#include "humdrum.h"

void createMidiFile(MidiFile& mfile, HumdrumFile& hfile);

int main(int argc, char** argv) {
   Options options;
   options.process(argc, argv);
   HumdrumFile hfile(options.getArg(1));
   hfile.analyzeRhythm();
   MidiFile mfile;
   createMidiFile(mfile, hfile);
   mfile.sortTracks();
   if (options.getArgCount() > 1) {
      mfile.write(options.getArg(2));
   } else {
      cout << mfile;
   }
   return 0;
}
```



MIDI files (2): createMidiFile

```
void createMidiFile(MidiFile& mfile, HumdrumFile& hfile) {
   int tpq = 120;
   mfile.setTicksPerQuarterNote(tpq);
   mfile.absoluteTime();  // inserted timestamps are not delta times
   mfile.allocateEvents(0, 100000);
   int i, j, k, pcount, midikey, starttick, endtick;
   Array<uchar> mididata(3);
   double starttime, duration;
   char buffer[1024] = {0};
   for (i=0; i<hfile.getNumLines(); i++) {
      if (!hfile[i].isData()) continue;
      for (j=0; j<hfile[i].getFieldCount(); j++) {
         if (strcmp("**kern", hfile[i].getExInterp(j)) != 0) continue;
         if (strcmp(".", hfile[i][j]) == 0) continue;
         pcount = hfile[i].getTokenCount(j);
         for (k=0; k<pcount; k++) {
            hfile[i].getToken(buffer, j, k);
            if (strchr(buffer, 'r') != NULL) continue; // rest
            if (strchr(buffer, '_') != NULL) continue; // tied note
            if (strchr(buffer, ']') != NULL) continue; // tied note
            if (strchr(buffer, '[') != NULL) {
               duration = hfile.getTiedDuration(i, j, k);
            } else duration = Convert::kernToDuration(buffer);
            starttime = hfile[i].getAbsBeat();
            if (duration == 0.0) {
               starttime -= 0.125; duration += 0.125;
            }
            starttick = int(starttime * tpq + 0.5);
            endtick = starttick + int(duration * tpq + 0.5);
            midikey = Convert::kernToMidiNoteNumber(buffer);
            mididata[0] = 0x90;  // note on: channel 1
            mididata[1] = midikey & 0x7f;
            mididata[2] = 64;
            mfile.addEvent(0, starttick, mididata);
            mididata[0] = 0x80;   // note off: channel 1
            mfile.addEvent(0, endtick, mididata);
}  }  }  }
```



MIDI files (3)

```
**kern
4c
8d
8c  8e
4d  4f
4g
*-
```

```
+++++++++++++++++++++++++++++++++++
Number of Tracks: 1
Time method: 1 (Absolute timing)
Divisions per Quarter Note: 120

Track 0   ++++++++++++++++++++++++

  0     0x90 60 64
120     0x80 60 64
120     0x90 62 64
180     0x80 62 64
180     0x90 60 64
180     0x90 64 64
240     0x80 60 64
240     0x80 64 64
240     0x90 62 64
240     0x90 65 64
360     0x80 62 64
360     0x80 65 64
360     0x90 67 64
480     0x80 67 64
+++++++++++++++++++++++++++++++++++
```