

## CS140 -- Lab C

- **CS140 -- Data Structures**
- **Fall-2006**
- **James S. Plank**
- **This file: <http://www.cs.utk.edu/~plank/plank/classes/cs140/Fall-2006/labs/labC/>**

Tue Nov 28 20:42:33 EST 2006

You are writing software to generate MP3 playlists. The program you are writing is called **playlist-maker**, and it takes filenames on the command line, plus one line of standard input. The filenames must end with the extension ".plist". Each file contains lists of songs in the following format:

```
Size Genre Artist Album Track Title
```

There is one song per line, and each line contains exactly six words:

- **Size:** This is the song file's size in bytes.
- **Genre:** This is the genre of the song.
- **Artist:** This is the artist or performer of the song.
- **Album:** This is the album from which the song was recorded.
- **Track:** This is the track number of the song on the album.
- **Title:** This is the title of the song.

None of the above will have whitespace in them. Therefore, you may use the fields library to read these files in.

Each song will also have a mood, which is specified by the playlist filename. Specifically, the part of the filename before the ".plist" is the mood of the file. So, "Upbeat.playlist" would have a mood of "upbeat." Care must be taken that if you specify a file by a long pathname, that the mood is appropriately constructed. For example, a playlist file whose name is "/home/plank/cs140/Fall-2006/labs/labC/Car.playlist" should have a mood of "Car".

Each song file may be located using the song's genre, artist, album, track and title. Specifically, the song's filename will be:

```
/Music Files/genre/artist/album/TN+title.mp3
```

where **genre**, **artist**, **album** and **title** are the obvious things, and **TN** is the track number padded to two digits, with a leading zero for two-digit numbers. So, the song specified by the first line of [Car.playlist](#):

```
3643436 Rock Who,_The Sell_Out 01 Armenia_City_In_The_Sky
```

corresponds to a song which may be found in the file:

```
/Music Files/Rock/Who,_The/Sell_Out/01+Armenia_City_In_The_Sky.mp3
```

**Playlist-maker's** first job should be to read in all the files listed on the command line, and to make sure that no duplicate songs have been specified. If there are any duplicate songs, it should flag it as an error and exit.

Next, **playlist-maker** should read exactly one line of standard input. This line should contain a number (decimal), and an **expression**. The number is a size in megabytes, and **expression** is a boolean expression in the following format:

- "**G string**" is an expression that is true if *string* occurs anywhere within a song's genre. Case matters (Think **strstr()**).
- "**R string**" is an expression that is true if *string* occurs anywhere within a song's artist. Case matters.
- "**A string**" is an expression that is true if *string* occurs anywhere within a song's album. Case matters.
- "**T string**" is an expression that is true if *string* occurs anywhere within a song's title. Case matters.
- "**M string**" is an expression that is true if *string* occurs anywhere within a song's mood. Case matters.
- If *e* is an expression, then "**(e)**" is also an expression, that is equivalent to *e*.
- If *e* is an expression, then "**!e**" is also an expression that is the negation of *e*.
- If *e1*, *e2*, ... *en* are all expressions, then "**(&& e1 e2 ... en)**" is also an expression that is equal to the boolean AND of all the expressions *e1* to *en*.
- If *e1*, *e2*, ... *en* are all expressions, then "**(|| e1 e2 ... en)**" is also an expression that is equal to the boolean OR of all the expressions *e1* to *en*.

So, the following are legal expressions:

- "**G Rock**" - Songs whose genre contains the string "Rock".
- "**( G Rock )**" - Songs whose genre contains the string "Rock".
- "**! G Rock**" - Songs whose genre does not contain the string "Rock".
- "**(&& G Rock )**" - Songs whose genre contains the string "Rock". Note, the AND and OR expressions do not have to have more than one subexpression.
- "**(&& G Rock T Blue )**" - Songs whose genre contains the string "Rock" and whose title contains the string "Blue".
- "**( && G Rock ! T Blue )**" - Songs whose genre contains the string "Rock" and whose title does not contain the string "Blue".
- "**(&& G Rock ( || T Blue T Red )**" - Songs whose genre contains the string "Rock" and whose title contains either the string "Blue" or the string "Red".
- "**! ( && G Rock ( || T Blue T Red )**" - All songs that do not match the expression above.
- "**( T )**" - Songs whose title contains a right parenthesis.

The following are not legal expressions:

- "**( G Rock**" - No right parenthesis.
- "**G Rock )**" - No left parenthesis.
- "**G Rock T Blue**" - Extra stuff after "G Rock".
- "**G Rock && T Blue**" - Improper format -- should be "**&& G Rock T Blue**".
- "**( ! )**" - "!" needs a subexpression.
- "**( && )**" - "&&" needs a subexpression.
- "**(G Rock)**" - The paren and the "G" need to be separated by whitespace.
- "**Genre Rock**" - This should be "**G Rock**".

Finally, **playlist-maker's** job is to emit a list of song files. Each song on the list should match the boolean expression, and the sum of the sizes of all the songs on the list should be close to, but less than or equal to the specified size.

On the last line, the total size, in megabytes, of all songs on the list, should be printed out to three decimal places. Remember that if *i* is a file's size in bytes, then the file's size is *i/1024/1024* megabytes.

I have example *plist* files in the lab's directory in the following files:

- [Upbeat.playlist](#) - Upbeat music.
- [Mellow.playlist](#) - Mellow, but not depressing music.
- [Pleasant.playlist](#) - Music that's pleasant music, but not quite as "feelgood" as the upbeat music.
- [Manic.playlist](#) - Maniacal music.
- [Car.playlist](#) - Music that is not in the above lists, but that I like to hear while driving.
- [Serious.playlist](#) - Music that is not in the above lists, but that I like to hear while working.
- [Depressing.playlist](#) - Just what it says.

Note that there are no duplicates in any of these files.

Here are some example sequences of calling `playlist-maker`. The following creates a list of roughly 30 MB of songs by the Dixie Dregs (the only artist with "Dreg" in it):

```
UNIX> playlist-maker *.plist
30 R Dregs
/Music Files/Rock/Dixie_Dregs/What_If/05+Ice_Cakes.mp3
/Music Files/Rock/Dixie_Dregs/Dregs_of_the_Earth/02+Pride_O'_The_Farm.mp3
/Music Files/Rock/Dixie_Dregs/Night_Of_The_Living_Dregs/08+Patchwork.mp3
/Music Files/Rock/Dixie_Dregs/Night_Of_The_Living_Dregs/04+Long_Slow_Distance.mp3
/Music Files/Rock/Dixie_Dregs/Night_Of_The_Living_Dregs/05+Night_of_the_Living_Dregs.mp3
/Music Files/Rock/Dixie_Dregs/What_If/01+Take_It_Off_The_Top.mp3
Total: 25.772 MB
UNIX>
```

If I wait a second and run it again, I'll get different songs:

```
UNIX> playlist-maker *.plist
30 R Dregs
/Music Files/Rock/Dixie_Dregs/Unsung_Heroes/02+Divided_We_Stand.mp3
/Music Files/Rock/Dixie_Dregs/Night_Of_The_Living_Dregs/08+Patchwork.mp3
/Music Files/Rock/Dixie_Dregs/Night_Of_The_Living_Dregs/01+Punk_Sandwich.mp3
/Music Files/Rock/Dixie_Dregs/Dregs_of_the_Earth/07+I'm_Freaking_Out.mp3
/Music Files/Rock/Dixie_Dregs/Dregs_of_the_Earth/06+Broad_Street_Strut.mp3
/Music Files/Rock/Dixie_Dregs/Dregs_of_the_Earth/02+Pride_O'_The_Farm.mp3
Total: 27.270 MB
UNIX>
```

Suppose I want a 50 MB playlist of songs by Jefferson Airplane that excludes their depressing stuff.

```
UNIX> playlist-maker *.plist
50 ( && R Jefferson_Airplane ! M Depress )
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/06+The_Last_Wall_Of_The_Castle.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/06+Law_Man.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/08+Third_Week_in_the_Chelsea.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/04+Martha.mp3
/Music Files/Rock/Jefferson_Airplane/Surrealistic_Pillow/01+She_Has_Funny_Cars.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/08+Watch_Her_Ride.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/03+Young_Girl_Sunday_Blues.mp3
/Music Files/Rock/Jefferson_Airplane/Surrealistic_Pillow/10+White_Rabbit.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/01+The_Ballad_Of_You_&_Me_&_Pooneil.mp3
/Music Files/Rock/Jefferson_Airplane/Surrealistic_Pillow/06+3-5_Of_A_Mile_In_10_Seconds.mp3
/Music Files/Rock/Jefferson_Airplane/Takes_Off/01+Blues_From_An_Airplane.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/04+Pretty_As_You_Feel.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/11+Won't_You_Try_-_Saturday_Afternoon.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/10+Two_Heads.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/11+War_Movie.mp3
Total: 46.602 MB
UNIX>
```

Or a 50 MB playlist of songs by either Jefferson Airplane or King Crimson that excludes manic and depressing stuff.

```
UNIX> playlist-maker *.plist
50 ( && ( || R Jefferson_Airplane R King_Crimson ) ! M Depress ! M Manic )
/Music Files/Rock/Jefferson_Airplane/Takes_Off/08+Don't_Slip_Away.mp3
/Music Files/Rock/Jefferson_Airplane/Surrealistic_Pillow/10+White_Rabbit.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/11+War_Movie.mp3
/Music Files/Rock/Jefferson_Airplane/Surrealistic_Pillow/11+Plastic_Fantastic_Lover.mp3
/Music Files/Rock/King_Crimson/Three_Of_A_Perfect_Pair/04+Man_With_An_Open_Heart.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/10+Thunk.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/04+Pretty_As_You_Feel.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/07+Rock_and_Roll_Island.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/04+Martha.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/02+Feel_So_Good.mp3
/Music Files/Rock/Jefferson_Airplane/After_Bathing_at_Baxter's/03+Young_Girl_Sunday_Blues.mp3
/Music Files/Rock/Jefferson_Airplane/Takes_Off/07+Run_Around.mp3
/Music Files/Rock/Jefferson_Airplane/Bark/08+Third_Week_in_the_Chelsea.mp3
/Music Files/Rock/Jefferson_Airplane/Takes_Off/04+It's_No_Secret.mp3
/Music Files/Rock/King_Crimson/In_The_Court_Of_The_Crimson_King/02+I_Talk_To_The_Wind.mp3
/Music Files/Rock/Jefferson_Airplane/Takes_Off/05+Tobacco_Road.mp3
Total: 48.998 MB
UNIX>
```

Or a 50 MB playlist that contains anything but the above songs:

```
UNIX> playlist-maker *.plist
50 ! ( && ( || R Jefferson_Airplane R King_Crimson ) ! M Depress ! M Manic )
/Music Files/Jazz/Spyro_Gyra/Morning_Dance/01+Morning_Dance.mp3
/Music Files/Classical/Odeon_Trio/Brähms_Piano_Trios_Volume_1/15+Brähms_Piano_Trio_in_Cm,_Opus_101,_3._Andante_grazioso.mp3
/Music Files/Classical/Richter,_Sviatoslav/Sviatoslav_Richter_-_Rachmaninoff/01+Rachmaninoff,_Piano_Concerto_#2,_Opus_18,_1._
/Music Files/Rock/Eurythmics/Sweet_Dreams/01+Love_Is_A_Stranger.mp3
/Music Files/Jazz/Machito_and_His_Orchestra/Bird_On_Verve_Volume_4/11+6-8.mp3
/Music Files/Jazz/Peterson,_Oscar/Tracks/02+Basin_Street_Blues.mp3
/Music Files/Rock/Tower_Of_Power/Back_To_Oakland/02+Don't_Change_Horses.mp3
/Music Files/Jazz/Jamal,_Ahmad/Ahmad's_Blues/01+Ahmad's_Blues.mp3
/Music Files/Rock/Wonder,_Stevie/Greatest_Hits/03+I_Was_Made_to_Love_Her.mp3
/Music Files/Jazz/Peterson,_Oscar/Trio_In_Transition/02+Younger_Than_Springtime.mp3
/Music Files/Rock/Van_Der_Graaf_Generator/The_Quiet_Zone_-_The_Pleasure_Dome/03+The_Siren_Song.mp3
/Music Files/Classical/Desormiere,_Roger_&_French_National_Symphony_Orchestra/Prokofiev,_Oranges_&_Kije/10+Prokofiev,_Suite_F
Total: 47.599 MB
UNIX>
```

## Strategy

First off, you'll need a random number generator. That's easy -- use `srand48()`, and `drand48()`. In the beginning of your program, call:

```
srand48(time(0));
```

This will set the "seed" to a random number generator so that it will generate different random numbers from the last time you called it (as long as you call it a second later than the last time you called it). Just call it once in your program. Then, whenever you need a random number, call **drand48()** (no arguments). This will return a random double between zero and one.

Now, write this in parts. The first part should read in the **plist** files and create a song **struct** for each song. That struct should have the fields **size**, **genre**, **mood**, **artist**, **album**, **title**, and **filename**. You should maintain a red-black tree of songs. This tree should have the song's filename as its key, and the song struct as its val. When you read a song, first create its **filename**, and check to see if it's already in the tree. If so, flag an error. Otherwise, insert it into the tree.

When you're done, traverse the tree and print out each song's filename. Test this program, and check it against my program **read-n-print** in the lab directory. Their outputs should be identical.

Next, write code to parse the expression on standard input. This is not easy. I used the following struct to hold an expression:

```
typedef struct enode {
    char type;
    char *string;
    struct enode *operand;
    Dllist operands;
} Expression;
```

The **type** is one of 'G', 'M', 'T', 'A', 'R', '!', '&' or '|'. If one of the first five, then its **string** is the string that is supposed to match to the given part of the song. If the type is '!', then **operand** is the expression to be negated. If the type is '&' or '|', then **operands** is a dllist of the subexpressions.

To parse the first line of standard input, I created a **dllist** that contains each word on the first line of standard input (with the exception of the first word). Then, I called a procedure:

```
Expression *parse_expression(Dllist input)
```

This returns the expression represented by **input**. **Parse\_expression()** is recursive. If the first string on **input** is a "G", "M", "A", "R", or "T", then it creates the proper expression, and deletes the first two nodes on **input**. Then it returns the expression. That's the easy case.

If the first string on **input** is a "!", then it creates an expression for it, deletes the first node of **input**, then sets the expression's **operand** field by calling **parse\_expression()** recursively.

If the first string on **input** is a "(", then it deletes that node, and checks the next node. If that node is "&&" or "||", then it recursively calls **parse\_expression()** and appends the result to the **operands** dllist, until it reaches a ")". If the "(" is not followed by "&&" or "||", then we assume that there is just one expression inside, so we recursively call **parse\_expression()** and return the result.

You may want to try drawing a flow chart for this. Think it through. Also, start small -- just implementing and testing the easy cases, and then moving on.

To test, I wrote a procedure called **print\_expression()** which does a preorder traversal of the given expression. Try my program **expression-parser()**, and test yours against it:

```
UNIX> expression-parser *.plist
50 ( && ( || R Jefferson_Airplane R King_Crimson ) ! M Depress ! M Manic )
&
|
R Jefferson_Airplane
R King_Crimson
!
M Depress
!
M Manic
UNIX> expression-parser *.plist
50 ( ( ( && G Rock T Roll ) ) )
&
G Rock
T Roll
UNIX>
```

Next, write the procedure:

```
int is_match(Song s, Expression *e);
```

This too will be recursive, and remember to use **strstr()** to implement "G", "M", "A", "R", and "T". Test it by calling it on every song in your song tree, and only printing out the matches. My program **expression-matcher** does this. Try it out, and compare it to your program.

Finally, modify your program so that instead of printing out the matches, you insert them into a new red-black tree, keyed on a random double. Then traverse this tree from the front, printing out song names until you either get to the end of the tree, or until the cumulative size of the songs is greater than the specified size.

Testing this last part is tricky, but I'm confident that you can do it. My **playlist-maker.c** is 268 lines long (no commenting, of course).