Homework 3: CLU

You should hand in a printout of your programs along with your answers to other problems, and please put your name on both. (If you do extra-credit problems, please write the names of the file(s) where the source code for your solutions can be found on your print-out.)

Edit your programs into a file `hw3.l` on Unix. (We will want to be able to see the `hw3.l` file in your directory.)

Test your programs using the `clu` command on Unix, which is the interpreter described in Appendix F.

Put your CLU code for the standard problems in `hw6.l`. Hand in a print-out of the code

1. (5 points) Write the function `reverse` which reverses the order of the top-level elements in a list. For lists, use the `List` cluster from the book.

2. (10 points) Do problem 2 on page 266 in Kamin's book (queues and level order traversal)

3. (10 points) Do problem 5 on page 267 of the text (rational numbers).

4. (10 points) Do one problems 7 on page 267 (windows).

5. (extra credit only) Do one or more of problems 3, 4, and 6 on page 267.

6. (10 points) Samuel Kamin remarks on the bottom of page 209 that the interpreter code is influenced by the idea of data abstraction. For this problem you will describe one such data abstraction found in the interpreter of chapter 6 (appendix G and appendix B) of the book. Describe (i) the abstract values of the objects, (ii) a vocabulary or notation for abstract values, (iii) and the operations on the objects, giving for each operation its name, type, and a English description of its effect, using the notations or vocabulary of (ii). The idea is to try to be as precise as you can in your descriptions. For examples, see the CLU lecture notes handout.

7. (extra credit only) Implement the data abstraction you described in the previous problem by a CLU cluster.

8. (25 points) This problem is to give you an idea of how data abstraction works in a larger program. The larger program in this case is part of what might be called a composer's or performer's assistant.

   Your program is to build on the code found in

   ```
   /home/cs342/public/src/clu
   ```

   In that directory is a file `vocal-outline.l`, which you should copy to your home directory, and edit to produce a solution to this problem. The following information is background for the problem, to make it self-contained. After that is a statement of the problem itself.

   Perhaps no one today would want to compose in the style of eighteenth century (European) classical music, but an aid to composing such music could easily be a help to performers and serious listeners. For example, one could compare a program's harmonization of a chorale (literally, a song) with Bach's, in order to better appreciate the subtle deviations from the "rules" in a masterwork.

To design a program for a domain in which you have little experience, you must learn some of the technical terms in that domain. This is especially important in object-oriented design, where the objects and concepts of the problem domain form the basis for a good solution to the problem. The following is an attempt to explain some of the terms from music theory relevant to this problem.

We will be considering four-part music, as for a choir of adult voices. There are four voices: *bass*, *tenor*, *alto*, and *soprano*. The lowest voice is the bass, next highest is the tenor; both of these are usually sung by men. Next highest is the alto, and highest of all is the soprano; these are usually sung by women. When a choir sings in four parts, each voice sings a different note (sound of a given frequency of vibration, which could be measured in Hertz).

Notes are named by their frequencies (also called pitches). Notes are ranked on a scale according to their frequencies. A note with a smaller frequency ranks lower on the scale than a note with a larger frequency. A note with frequency f is said to be an *octave below* a note with frequency $2f$. Notes whose pitches differ only by an octave sound very similar.

Musicians rarely think in terms of frequencies measured in Hertz. Instead, tonal music is based on a scale of seven pitches (the diatonic scale). The seven pitches are called *scale degrees* and are given numbers from 1 to 7. (These are traditionally written with capital Roman numerals.) These scale degree numbers are used regardless of the particular key (which is the pitch of the note numbered 1) and regardless of the type of scale (e.g., major or natural minor). This abstraction allows all keys and scales to be treated similarly, in effect by a linear transformation (or transposition).

It is useful, for this homework, to depart from traditional music theory slightly and allow scale degrees that are larger than 7. If the pitch of the note numbered 1 is f, the scale degree of the note with pitch $2f$ will be called 8. The next higher scale degree will be 9, and so on. For many purposes in music theory octave differences are disregarded. Octave differences in pitch translate to adding multiples of 7 to the scale degree. So from a mathematical point of view, much of music theory is done modulo 8 plus 1. In what follows, the term *note* will be used when octave differences are disregarded, and the term scale degree will be used when an absolute scale of pitches is needed. For example, the scale degree 12 is regarded as the same note as scale degree 5.

The difference in scale degrees between d and $d + 2$ is called a *third*. The difference in scale degrees between d and $d + 4$ is called a *fifth*. The difference in scale degrees between d and $d + 7$ is called a *octave*.

Three or more notes sung at the same time make a *chord*. We will consider only chords in which four notes are sung at the same time, one by each voice. Furthermore, we will only consider certain kinds of consonant chords: triads with one note doubled.

A *triad in root position* consists of three notes, whose scale degrees are d, $d+2$, and $d+4$; for example, in the triple of scale-degrees, $(3, 5, 14)$, the lowest note is $d = 3$, the next highest is $d + 2 = 5$, and the highest is $d = 14$. Mathematically, a triple of scale degrees (d_1, d_3, d_5) is a triad in root position if and only if $d_1 < d_2$, $d_2 < d_3$, $d_2 = d_1 + 2 + k \times 7$ and $d_3 = d_1 + 4 + k \times 7$ (for some integer k). Note that octave differences are ignored. Triads need not appear in root position; for example, the triple of scale degrees $(5, 8, 10)$ is also a triad, as is $(4, 16, 26)$. So a triad is a set of three scale degrees, $\{d, d + 2 + k \times 7, d + 4 + k \times 7\}$. The note d in a triad

$\{d, d+2+k\times 7, d+4+k\times 7\}$ is called the *root* of the triad.

A triad with one note doubled has 4 notes, enough for each voice in a choir. That is, a triad with one note doubled can be represented by a tuple of 4 scale degrees, (b, t, a, s). It is convenient to arrange the scale degrees in a triad so that $b < t < a < s$, thus b represents the scale degree of the tone the basses sing, t the tenor, a the alto, and s the soprano. A *4-note chord in root position* is a tuple of 4 scale degrees, (b, t, a, s), such that $b < t < a < s$, and for each $d \in \{0, 2, 4\}$, there is exactly one $v \in \{t, a, s\}$, $v = b + d + k \times 7$, where k is an integer. In such a chord, b is the *root*. Note that the tenor, alto, and soprano all sing different notes: for example, the difference in scale degree between the tenor and alto should not be a multiple of 7 (an octave).

Implement the 4-part voice leading problem as described below in CLU (using the code described above).

The program's input is an *initial chord* and a *base line*. A *bass line* is a list of notes. Write a program that takes as its input a sequence (as represented by consecutive integers) of scale degrees, (b_i), and which outputs sequence of 4-note chords in root position, (c_i), of the same length, such that each b_i is the root of c_i.

The program's output is a list of chords such that: the bass note of the ith output chord is the ith note of the bass line, the first chord of the output is the initial chord of the input, and the following rules are satisfied:

(a) To make the music more melodic (i.e., less jumpy), for each voice (tenor, alto, soprano), the (absolute value of the) difference in scale degrees from the note that voice sings in chord c_{i-1} to c_i should be as small as possible. For example, if c_{i-1} is $(1, 10, 12, 15)$, and the bass of c_i is to be 5, then c_i must be $(5, 9, 12, 14)$. This rule applies globally; that is, if one had started by selecting 16 for the soprano of c_i, then after selecting 12 for the soprano, the tenor of c_i would have to be 7, which is not as close as 10.

(b) To break ties in the above rule (i.e., two notes equally close that are possible), the sign of the difference should be opposite to that for the bass.

(c) Be careful not to let the voices cross! For example, the tenor should always be higher than the bass, and lower than the alto.

9. (extra credit only) Do one or more of the problems in section 6.9.3 of the text (program verification).

10. (extra credit only) Do (one or more of) the problems 8 or 9 in section 6.9.2 of Kamin's book. You will find the source for the clu interpreter in

```
/home/cs342/public/src/chap.6/clu.p
```

Make a copy using the Unix `cp` command and then edit your copy. Turn in differences (using the Unix command `diff`) as before.