CSC112 Fall 2010
Fundamentals of Computer Science
Lab 10

In this week's lab, we will re-visit working with sounds, but this time we will be using C++ classes and overloaded operators.

Write a C++ class to represent a sound. Sounds are stored digitally as an array of signed short integers. For the purpose of this lab, we will assume all samples will be played at the compact disc sampling rate of 44,100 samples per second. Class `sound` should represent the sound using two pieces of private data: `n` is the number of samples, and `the_samples` is a dynamically allocated array of `short int` of length n. Your class should start:

```
class sound {
 private:
    int n ;
    short int * the_samples ;
 public:
       .
       .
       .
```

Your class must support the following operators:

1. Overload the `+` operator so that it will mix two sounds. I.e., `s3 = s1 + s2 ;` averages the two sounds so that they will be heard simultaneously.
2. The operator `+=` must work as expected.
3. Overload the `&` operator so that it will concatenate two sounds. I.e., `s3 = s1 & s2 ;` will result in a new sound `s3` which consists of sound `s1` followed by sound `s2`.
4. Overload the `*` operator so that it will accept a sound s and an integer k. The resulting sound should consist of s repeated k times. I.e., multiplication by an integer acts like repeated addition.
5. The operator `*=` should work as expected.

Your class must support the following member functions:

1. `sound()` The constructor taking no parameters creates an empty sound, with `n==0`, and `the_samples == NULL`.
2. `~sound()` The destructor taking no parameters frees the memory allocated to a sound. Beware of deleting `NULL` pointers. I.e., when an empty sound object goes out of scope.
3. `void play()` Write the sound to a temporary file, use the `system` function and the `play` command to play the sound. When done, delete the temporary file (see man unlink).
4. `void sine_wave( double frequency, double duration )` Create a sound consisting of a pure sine wave. Input parameter `frequency` is in cycles per second. Input parameter `duration` is in seconds.

5. `void square_wave( double frequency, double duration, double pulse_width )` Create a sound consisting of a square wave of the given frequency and duration. The input parameter `pulse_width` should be a floating point number between 0 and 1. It represents the fraction of time that the wave is positive. If omitted, `pulse_width` has the default value of 0.5.

6. `void save( char * filename )` Writes a file representing a sound

7. `void read( char * filename )` Reads a file representing a sound

8. `sound & volume( double v, bool & clip )` Input parameter `v` is a positive real number between 0 and 11.[1] Function `volume` adjusts the volume of a sound by a factor of `v`. If `v == 1` then the volume is unchanged. The pass by reference parameter `clip` is set to true if an increase in volume causes **clipping**, i.e., when the sound sample value is multiplied by `v`, the result is greater (in magnitude) than can be represented in a 16-bit short int. If no clipping occurred, then the pass-by-reference variable `clip` is set to false.

   This function returns a reference to a new sound. The original sound object is unchanged.

Your class must work with the main program supplied for this lab.

See:

```
http://menehune.opt.wfu.edu/CSC112
```

to download the main program.

Create a directory named Lab10. Keep all of your source and compiled programs in the directory Lab10.

Turn in: Change to the directory containing the sub-directory "Lab10". Create a file named "lab10.tar" using the command:

```
tar cf lab10.tar Lab10
```

Upload the file "lab10.tar" to your account on telesto.

[1]

Nigel Tufnel: The numbers all go to eleven. Look, right across the board, eleven, eleven, eleven and...
Marty DiBergi: Oh, I see. And most amps go up to ten?
Nigel Tufnel: Exactly.
Marty DiBergi: Does that mean it's louder? Is it any louder?
Nigel Tufnel: Well, it's one louder, isn't it? It's not ten. You see, most blokes, you know, will be playing at ten. You're on ten here, all the way up, all the way up, all the way up, you're on ten on your guitar. Where can you go from there? Where?
Marty DiBergi: I don't know.
Nigel Tufnel: Nowhere. Exactly. What we do is, if we need that extra push over the cliff, you know what we do?
Marty DiBergi: Put it up to eleven.
Nigel Tufnel: Eleven. Exactly. One louder.
Marty DiBergi: Why don't you just make ten louder and make ten be the top number and make that a little louder?
Nigel Tufnel: [pause] These go to eleven.