Assignment # 1

Dynamic 1-D Arrays

Submission Dead Line: Sunday 6/4/2021

Important Notes

- Use meaningful variable names
- YOU MUST DEALLOCATE ALL MEMORY PROPERLY, YOUR CODE SHOULD NOT HAVE ANY **MEMORY LEAKS** OR **DANGLING POINTERS**.
- Indent your program so that statements inside a block can be distinguished from another block
- Use const type instead of using hard coded numbers
- You ARE NOT ALLOWED to use break, continue and goto
- You ARE NOT ALLOWED to use global variables
- You ARE NOT ALLOWED to use more than one return statement in a function
- All functions that compute values should have NO cin and cout. Exchange all data via parameters.
- Functions that compute values should be kept separate from those that draw on screen
- Spaghetti code and confusing code is not acceptable
- Comment your code properly

Question: Write a C++ program for processing of voice data read from (.wav) files.

1. **Allocate Memory:** Write a function that allocates memory for a dynamic array of any given size.
   ```
   void allocateArray(unsigned char *& arr, int size);
   ```
2. **Deallocate Memory:** Write a function to deallocate the Array. All memory must be deallocated properly at end of program.
   ```
   void deallocateArray(unsigned char *& arr);
   ```
3. **Print:** write a function to print the contents of array of any given size.
4. **Doubles an array:** Write a function that doubles an array by duplicating items in the array and returns doubled array. This function will not change the original array, make careful selection of function parameters.
   For example:
   Original array: {4,2,1,9,1,3,6}
   Doubled array: {4,4,2,2,1,1,9,9,1,1,3,3,6,6}

   The above is a special case of a procedure called up sampling, where you are increasing the number of samples in a signal.

5. **Shrink an array:** Write a function that reduces an array's size to half by deleting every other item of the array and returns shirked array. This function will not change the original array, make careful selection of function parameters.

   For example:
   Original array: {4,2,1,9,1,3,6}
   Halved array: {4,1,1,6}

   The above is a special case of a procedure called down sampling of a signal, which takes a selected set of values from the original array and reduce the samples to half.

6. **Fill with Mean:** Write a function **FillWithMean** that takes two arrays, **in** and **out**, and a positive number **N** as parameters.

   The function then fills each index **i** of the array out with the mean/average of **N*2+1** value in the neighborhood of index **i** from the array in. That is, it will take the mean of the **ith** number and N numbers to its left and N numbers to its right. You also need to take care of the boundaries as shown in the examples below. The input and output arrays should be different. This procedure is called the method of moving averages.

EXAMPLE 1 with N = 1

index	0	1	2	3	4	5
in	4	5	3	6	1	3
out	mean(4,5) =9/2	mean(4,5,3) =4	mean(5,3,6) =14/3	mean(3,6,1) =10/3	mean(6,1,3) =10/3	mean(1,3) =2

EXAMPLE 2 with N = 2

index	0	1	2	3	4	5
in	4	5	3	6	1	3
out	mean(4,5,3) =4	mean(4,5,3,6) =18/4	mean(4,5,3,6,1) =19/5	mean(5,3,6,1,3) =18/5	mean(3,6,1,3) =13/4	mean(6,1,3) =10/3

About audio wav files:

Wav files are nothing but sequences of arrays, containing sound samples. The most basic audio files are 'mono', which means only one channel with 8000 samples in one second, each sample being one byte long. You are provided with two basic audio files reading and writing functions.

The **readWavFile** function will read the sound values in an array and also return the sampling rate (bytes per second). Increasing the sampling rate of audio will improve the sound quality. Decreasing the sampling rate will decrease the sound quality. The wav file functions provided will give you a basic idea about audio storage and manipulation.

```
readWavFile("sallimono.wav", arr, size, samplingRate);
//will read the audio file

writeWavFile("sallimono.wav", arr, size, samplingRate);
//will write the audio file
```

Notice:

1. the data is read into an array of type unsigned char, which means each value of audio file is one byte long an unsigned number.
2. The function readWavFile will also place the bytes read in the variable size. Size is an input output parameter.
   a. If the input size is more than audio size then size is changed to actual audio size.
   b. If size is less than the audio file's size then only the length amount of bytes are read in the array.
3. The function also returns the sampling rate, which indicates how many audio values were collected in one second. You can print this value to see what it is.
4. You can also print the array to see the values stored in it.

7. **Read from File:** Write a function **readFromFile** which takes file name from user as input and then calls `readWavFile` function to read the contents of the array from the file.

8. **Up sample an audio file:** Write a function **upSampleAudio** which up samples already read sound file by calling double array function and then write data back in a new wav file by calling function `writeWavFile` function. After writing the new array, listen to the new audio file and see what your observations are. Notice how the pitch is changed!

9. **Down sample an audio file:** Write a function **downSampleAudio** which down samples already read wav file by calling shrink array function and then write data back in the new wav file by calling function `writeWavFile`. After writing the new array, listen to the new audio file and see what your observations are.

10. **Apply moving average filter to the audio array:** Write a function **movingAverageFilter** which will create the copy of original wave file and will apply moving average filter by calling fill with mean function and then write data back in wav file by calling function `writeWavFile`.
    Listen to the new audio file and note your observations. Repeat this for different values of N like, 1,3, 5, 15, 100.

11. **Mix two audio signals together:** Write a function called **mergeArray**, which takes as input two arrays and their sizes and returns a third array merged with values of first two arrays. For example:
    Array1: {1,2,3,4,5,6,7,8}
    Array2: {10,29,50}
    Mergedarray: {1,10,2,29,3,50,4,5,6,7,8}

    Read dhani.wav and salli.wav and construct a third array which has mixed data of both arrays (you need to call mergeArray). Write the resulting in a wav file and listen to the wav file. When saving the wav file, save its two copies one with salli's sampling rate and other with dhani's sampling rate. What are your observations after listning?

12. **Menu:** For all the parts given above make a menu function to access and test each function. You can repeat this for other music files (.wav) also.

HAPPY PROGRAMMING