CSCI 2270 – Data Structures

Assignment 2

DYNAMICALLY ALLOCATED ARRAYS

OBJECTIVES

1. Read a file with unknown size and store its contents in a dynamically allocated array
2. Store, search and iterate through data in an array of structs
3. Use array doubling via dynamic memory to increase the size of the array

Instructions

In this assignment, we will write a program to analyze the word frequency of a plain text document. As the number of words in the document may not be known a priori, we will perform array doubling on a dynamically allocated array to store the information as it arrives.

Please read all the directions *before* writing code, as this write-up contains specific requirements on how the code must be written.

Problem

Overview:

There is a file on github:

(1) ***movies.txt*** - contains the name of the movies, user ratings and MPA film ratings in tab separated format. (ex. `Cinderella'\t'0.72'\t'G`). There are possibly multiple ratings for the same movie. They will each be represented in a new line.

Your program must take three command-line arguments in the following order:

(1) ***the name of the txt file to be read and analyzed (movies.txt)***
(2) ***the MPA rating to be ignored (PG-13)***
(3) ***The number of movies from the top of the ranking***

Your program will read the text from the first file and store all the unique movies encountered in a dynamically doubling array. After necessary calculation(s), the program must print the following information:

- The number of times array doubling was required to store all the movies
- The number of unique movies in the file (how many times the same movie appeared)
- The total movie count of the file
- After calculating the average ratings of each movie and storing it in an array in the decreasing order of **user rating**, print the **N** most highly rated movies along with their average ratings (**Rounding to the 3rd decimal place**) from the top of the resultant array.

For example, running your program with the command:

```
./run_app_1 ../movies.txt PG-13 5
```

would print the 5 movies starting from rank 1 (of the sorted-by-rating array), i.e. your program must print the 1st-5th most highly rated movies, along with their average ratings. Keep in mind that these movies must be distinct ones that are not duplicated by each other.

A sample run will look as follows:

```
Array doubled: 7
Distinct # of movies except PG-13: 1049
Total # of movies excluding PG-13 ratings: 5893
Movie Ratings
---------------------------------------
0.965 The Godfather Part II
0.955 The Godfather
0.955 E.T. the Extra-Terrestrial
0.950 Jaws
0.950 Schindler's List
```

Specifications:

1. Use an array of structs to store the movies and their counts

You will store each unique movie and its count (the number of times it occurs in the document) in an **array of structs**. As the number of unique movies is not known ahead of time, the array of structs must be **dynamically sized**. The struct must be defined as follows:

```
struct movieRecord
{
	string movieName;
	int movieCount;
	float avgUserRating;
};
```

As you read movies from the file and store them in an array, you will calculate the running average of user ratings for that movie. For example- When you find 'avengers' for the first time with user rating 3.8 you will store the movie with *movieCount* 1 and *avgUserRating* as 3.8. Now after sometime you again find 'avengers' with user rating as 4. So now you will update the record for 'avengers' with *movieCount* as 2 and *avgUserRating* as 3.9. After a few more lines you again encounter 'avengers' with user rating say 3.5. Now you will update *movieCount* with 3 and update the *avgUserRating* as 3.76666. So *avgUserRating* is actually recording the running average of the user ratings for a particular movie. You can update it as following way-

*avgUserRating = (avgUserRating * prev_count + userrating)/current_count*

Where `prev_count` is the number of occurrences of this movie prior to the current encounter and `current_count` is the updated count after this current occurrence. And userrating is the user rating in the current occurrence. So for instance in the example of 3rd occurrence of 'avengers' with user rating 3.5 we already had avengers in our array with *movieCount* with 2(*prev_count)* and *avgUserRating* as 3.9. After the 3rd occurrence *movieCount* became 3 (*current_count)*. So the

avgUserRating = (3.9 * 2 + 3.5) / 3 = 3.76666

So in context of our structure definition we can calculate this as-

```
distinctMovies[ind].avgUserRating = (float)
((distinctMovies[ind].movieCount-1)*distinctMovies[ind].avgUserRating + current_userrating)/
distinctMovies[ind].movieCount;
```

2. Use the array-doubling algorithm to increase the size of your array

Your array will need to grow to fit the number of movies in the file. **Start with an array size of 10**, and double the size whenever the array runs out of free space. You will need to allocate your array dynamically and copy values from the old array to the new array. (Array-doubling algorithm must be implemented in the *main()* function).

Note: Don't use the built-in **std::vector** class. This will result in a loss of points. You're writing code that emulates the vector container's dynamic resizing functionality.

3. Ignore the records with the MPA rating(ex. PG-13) which is given through the command-line argument.

If you type in 'PG-13' for the second argument, your result should exclude all the records with PG-13 MPA rating.

4. Take three command-line arguments

Your program must take three command-line arguments

1. the name of the CSV file to be read and analyzed
2. The MPA rating to be excluded: this is the MPA rating (G, PG, PG-13, R, NC-17)
3. an integer N: this is the number of movies to be printed

5. Output the *N* highest rated movies

Your program must print out **N** highest user rated movies - not including the movies with a given MPA rating(to be excluded) - starting from the top. This assumes that the array you are indexing on is sorted by ratings (descending). **If two movies have the same ratings, list them alphabetically.**

You may assume that N will always be valid values for a given example, provided that array doubling was performed correctly.

6. **Format your final output this way:**

```
 Array doubled: <Number of times the array was doubled>
 Distinct # of movies except PG-13: <Distinct number of movies>
 Total # of movies excluding PG-13 ratings: <Total number of movies
before merging duplicates>
 Movie Ratings
 ---------------------------------------
 <1st hightest rating> <Corresponding movie>
 <2nd hightest rating> <Corresponding movie>
 <3rd hightest rating> <Corresponding movie>
 <4th hightest rating> <Corresponding movie>
 <5th hightest rating> <Corresponding movie>
```

For example, using the command:

```
./run_app_1 ../movies.txt R 7
```

Output:

```
Array doubled: 7
Distinct # of movies except R: 838
Total # of movies excluding R ratings: 5358
Movie Ratings
---------------------------------------
0.955 E.T. the Extra-Terrestrial
0.950 Jaws
0.945 Dr. Strangelove
0.945 The Bridge on the River Kwai
0.940 Psycho
0.940 I Am Not Your Negro
0.935 Toy Story 3
```

7. **You must include the following functions (they will be tested by the INGInious):**
   a. **../app_1/main function**
      i. If the correct number of command-line arguments is not passed, print the below statement as-is and exit the program

```
cout << "Usage: ./run_app_1 <inputfilename> <MPA rating> <N>" << endl;
```

ii. Array-doubling function must be called in the main() function

iii. Read movies from the provided TXT file (ex: ***movie.txt***) and store all unique movies that are not with the given <MPA rating>.

1. Create a dynamic **`movieRecord`** array of size 10
2. Add movies to the array (not for the movies with the given MPA rating)
3. Keep track of the number of times the **`movieRecord`** array is doubled and the number of unique movies

b. checkMovieToBeIncluded function

```
bool checkMovieToBeIncluded(string movieMpaRating, string
mpaRatingToBeExcluded);
```

This function checks whether a movie with **`the given MPA rating`** (from file data) should be included or not.
Ex: if Cinderella movie has PG-13 rating, and the rating to be excluded (Taken from command line args) is PG-13, the function should return "false" as it should not be added to the struct array.

c. doubleArray function

```
Void doubleArray(movieRecord *&movieArray, int &arrCapacity);
```

This function should double the given array by dynamically creating a new movieRecord array of twice the existing capacity, copy the contents of the movieArray to the new array. Make sure to handle memory leaks by appropriate deletions.

d. getTotalMoviesCount function

```
int getTotalMoviesCount(movieRecord* distinctMovies, int
length);
```

This function must compute the total number of movies in the entire document by summing up the counts of the individual unique movies. The function must return this sum.

e. sortArray function

```
void sortArray(movieRecord* distinctMovies, int length);
```

This function must sort the **`distinctMovies`** array (which contains **`length`** initialized elements) in descending order by their average ratings, such that the most highly rated movies are sorted to the beginning. The function does not return anything.

Additionally, if a subset of movies has the same rating then they should also be sorted alphabetically in ascending order.

> ***Note:*** You should NOT use the built-in **`sort`** function provided by the **`<algorithm>`** header. However, you may write your own implementation of a sorting algorithm, such as Bubble Sort. Feel free to refer to the pseudocode for bubble sort that was given in Assignment 1.

f. printTopNMovies function

```
void printTopNMovies(movieRecord* distinctMovies, int N);
```

This function must print highest user rated *N* movies from the **sorted** array of **`distinctMovies`**. These *N* movies must be printed with their average ratings **round to 3rd decimal places** *(refer Appendix 1)*. The exact output format is given below. The function does not return anything.

The format for printing the movies should be as follows:

cout<<**RATING**<<" "<<**Movie name**<<endl;

The average rating of a movie at position ***ind*** in the array is computed using the formula assuming the `movieCount` has been adjusted to encounter the recent occurrence (refer to the example given at top of page 2) : ***(Don't forget to cast to float!)***

```
distinctMovies[ind].avgUserRating = (float)
((distinctMovies[ind].movieCount-1)*distinctMovies[ind].avgUserRating +
current_userrating)/ distinctMovies[ind].movieCount;
```

APPENDIX 1: Printing *Z* decimal places

In order to print *Z* number of decimal places in C++, you need to include the **`<iomanip>`** header.

You can then format your code as follows:

```
int Z = 3;
cout << fixed << setprecision(Z);
cout << 123.45 << endl;              // Prints 123.45000
cout << 0.01234567 << endl;          // Prints 0.01235
```

CSCI 2270 – Data Structures

APPENDIX 2: Reading words from a text file

Use the following C++ snippet to read words from a file:

```
#include <fstream>

ifstream inStream;        // stream for reading in file
inStream.open(filename); // open the file

string movie;
while ( getline(inStream, movie) )
{
        // process the movie by tab separated values, by using
istringstream or similar constructs.
}
inStream.close();         // close the file
```

APPENDIX 3: MPA Film Rating