CSCI 2270 – Data Structures

Assignment 2

DYNAMICALLY ALLOCATED ARRAYS

OBJECTIVES

1. Read a file with an 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 you are going to create a dynamically allocated array, populate it with information, and then, as required, repeatedly double the array during run-time. Please read all the directions *before* writing code, as this write-up contains specific requirements for the code.

Problem

Overview:
You have been provided with a file: **cases.txt**, which comprises records that represent covid-19 cases. It contains information about the affected person's name, virus variant, location, and age. Each line is a comma-separated list of the above-mentioned information. An example of a line is `Bob,Delta,Boulder,34.`

We want to read the data, store it in a data structure, and query from it. A query will be of the type: *Who are the people in a particular **location,** affected by a particular covid-19 **variant**, and who belong to a specific **age** group.*

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

1. The name of the txt file to be read and analyzed (e.g., cases.txt)
2. The location (e.g., Boulder)
3. The variant of the virus (e.g., Delta)
4. The lower limit (included) of the age group (e.g., 20)
5. The upper limit (included) of the age group (e.g., 40)

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

- The number of times array doubling was required to store all the cases.
- The total number of cases returned after the query.

- The cases which are returned after the query is hit. Note that the cases will have to be sorted in **ascending order of the person's age**.

  For example, running your program with the command:

```
./run_app ../cases.txt Boulder Delta 20 25
```

  would print the cases where the individuals were affected by the Delta variant, who live in Boulder and are in the age group 20-25, in increasing order of their age.

A sample run will look as follows:

```
Array doubled: 1
Total number of cases returned after the query: 3
Queried Cases
--------------------------------------
Bob 25
Alice 26
James 28
```

Specifications:

1. **Use an array of structs to store the cases.** You will store each unique case in an **array of structs**. As the number of cases is not known ahead of time, the array of structs must be **dynamically sized**. The struct must be defined as follows:

```
struct CovidCase
{
        string name;
        string location;
        string variant;
        int age;
};
```

As you read the file, store the information of a line (i.e. name, variant, location, and age) in a struct variable of the type **`CovidCase.`** This variable will be appended to the dynamically sized array of structs. **One must note that only the cases which satisfy the query parameters must be appended to the array.**

CSCI 2270 – Data Structures

2. **Use the array-doubling algorithm to increase the size of your array.** Your array will need to grow to fit the number of cases 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 dynamically allocate your array and copy values from the old array to the new one.

> ***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 that do not satisfy the query parameters (i.e., location, age, variant) given through the command-line argument.** If you type the location as Boulder, the variant as Delta, and the age range as 20 to 30, then do not append the records where the person is not from Boulder or is not affected by Delta, or belongs to an age less than 20 or more than 30. Note that one is **not supposed to exclude** a person with an age of exactly 20 or 30.

4. **Output the records for a query.** Your program must print out the names and the ages of the patients, which satisfy the query parameters in **increasing order** of their age. If there is a tie, you can break that by using the person's name (i.e., alphabetically). For example, if there are two records (Bob and Alice) of the same age (let's say 25), you must print Alice's record first.

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

```
Array doubled: <Number of times the array was doubled>
Total number of cases returned after the query: <corresponding value>
Queried Cases
--------------------------------------
<Name of the first person> <Corresponding age>
<Name of the second person> <Corresponding age>
…
```

For example, using the command:

```
./run_app ../cases.txt Eldora Delta 20 25
```

Output:

```
Array doubled: 1
Total number of cases returned after the query: 15
Queried Cases
--------------------------------------
Anjela Norman 20
Dartanian Starr 20
Jenne Ryan 20
Julissa McIntyre 20
Magic Archer 20
Terrian Brock 20
Brad Dixon 21
Joi Parker 21
Shamar Stafford 21
Carlis Benton 22
Latiya Williamson 22
Jamira Pierce 23
Lexie Blackburn 23
Danetta Mueller 24
Yury Ferguson 24
```

6. **You must include the following functions (they will be tested in INGInious):**
   a. **../app/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 <inputfilename> <Query_Location>
<Query_Variant> <Query_Start_Age> <Query_End_Age>" << endl;
```

      ii. Open the input file (ex: ***cases.txt***) here.
      iii. Start with a dynamic **`CovidCase`** array of size 10
      iv. You need to read the command line arguments in the main function and pass them to the **`parseFile`** function as an array of query parameters.
      v. You need to print the number of times the array has been doubled and the total number of cases returned by the query. (See the format provided in **5**).
      vi. In the end, call the **printQueriedCases** function to print all the queried cases.

   b. **parseFile function**

```
void parseFile(ifstream& input,string queryParams[], CovidCase
*&cases, int &arrCapacity,  int &recordIdx, int
&doublingCounter);
```

This function parses the given **input file** line by line and takes in the **query parameters** passed from the command line. We use these parameters along with the lines read from

the input and store all the Covid Cases that satisfy the query conditions (Location, Variant and Age Range). Keep track of the number of times the **`CovidCase`** array is doubled.

c. isCaseQueried function

```
bool isCaseQueried(CovidCase case, string queryLocation, string queryVariant, int startAge, int endAge);
```

This function checks whether a single case with the given **location** and **variant** belongs to the given **age range**.

For example, let's say you have an entry of **Bob**, aged **25**, living in **Boulder** and had the **Delta** variant. The **command line arguments** ask you to get cases in Boulder with Delta variant between the **age range from 20 to 30**; then this function should return True for this entry since it should be added to the array.

d. resizeArr function

```
void resizeArr(CovidCase *&cases, int *arraySize);
```

This function should resize the given array by dynamically creating a new ***CovidCase*** array of twice the existing capacity, copy the contents of the ***cases*** to the new array. Make sure to handle memory leaks by appropriate deletions.

e. addCase function

```
void addCase(CovidCase *&cases, CovidCase covidCase, int &arrCapacity,int &recordIdx, int &doublingCounter);
```

This function must add the entry which returned True for ***isCaseQueried*** function to the array cases. This function will not return anything but will add the filtered entry to the array to store it.

f. sortArray function

```
void sortCases(CovidCase* cases, int length);
```

This function must sort the cases array (which contains **`length`** initialized elements) in ascending order of the ages of the person, such that the person with the smallest age is sorted to the beginning. **Additionally**, if a subset of entries has the same age then they should also be sorted alphabetically by name in ascending order. The function does not return anything.

CSCI 2270 – Data Structures

> ***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.

g. printQueriedCases function

```
void printQueriedCases(CovidCase* cases, int numOfRecords);
```

This function must print the name and age of the person sorted (in ascending order) by their ages from the **cases** array. The exact output format is given below. The function does not return anything.

The format for printing the **queried cases** should be as follows:

```
cout<<"Queried Cases\n--------------------------------------"<<endl;

cout<<Name<<" "<<Age<<endl;
```

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
```

APPENDIX 2: Reading cases from a text file

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

```
#include <fstream>

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

string case;
while ( getline(inStream, case) )
{
```

```
        // process the case by tab separated values, by using
istringstream or similar constructs.
}
inStream.close();        // close the file
```