Nikolas Iliopoulos 1115201800332

1 Exercise System Programming

There are comments in the code.

Compile & Run

Makefile is included. Use the command **make** to compile the source code.

To run the application use: **./vaccineMonitor -c inputFile.txt -b BloomSize**

> *Note* that the order of the arguments doesn't matter. ./vaccineMonitor -b BloomSize -c inputFile.txt

---

Error Handling & Checking

- Syntax of Records
  - The line must have 7 or 8 words
  - id is only numbers
  - 0 <= id <= 9999
  - age is only numbers
  - 1 <= age <= 120
  - 7 word must be either YES or NO
  - If there are 7 words total the seventh word must be NO
  - If there are 8 words total the seventh word must be YES
  - The date if exists must have the format dd-mm-yyyy
  - 1 <= dd <= 30 & 1 <= mm <= 12 & 1950 <= yyyy <= 2050
- Bad dupicates
  - Citizens with same id must match the credentials firstname lastname country age in order to be a valid duplicate of that citizen
  - Citizens that have valid credentials must also have a different virus name that the already inserted citizen has.

Structure & Classes

VaccineMonitor

The files **vaccineMonitor.cpp** and **vaccineMonitor.h** contains the *vaccineMonitor class*.

This class contain methods for each command, reading records from files and and the menu. The data members are headers for the tree of citizens, the list of countries, list of viruses, list of bloom filters and list of skiplists.

```
class vaccineMonitor
{
public:
    //Methods
private:
    treeNode* tree;
    stringList* countryList;
    stringList* virusList;
    bloomFilterList* bloomList;
    skipList_List* skiplist;
};
```

Citizen Record

The files **citizenRecords/citizen.cpp** and **citizenRecords/citizen.h** contains the *citizenRecord class* and the *listStatus class*. Every citizen has its own list where the viruses information is beeing stored.

> *Note* that in citizenRecord the country is a node of type stringList which is a list that stores strings. This class is used to prevent data duplications for countries and viruses as the same country or virus name has to be referenced many times.

```
class citizenRecord
{
public:
    //Methods
private:
    int citizenID;
    string firstName;
    string lastName;
    stringList *country;
    int age;
    listStatus *status;
};
```

```
class listStatus
{
public:
    //Methods
private:
    stringList *virusName;
    char status; // 'y' stands for "YES", 'n' for "NO"
    date dateVaccinated;
    listStatus *next;
};
```

stringList

The files **DataStructures\linkedList\linkedListString.cpp** and **DataStructures\linkedList\linkedListString.h** contains the *stringList class*.

This class is used to create the list of countries and list of virus names.

```
class stringList
{
public:
    //Methods
private:
    string data;
    stringList *next;
};
```

Date

The files **DataStructures\date\date.cpp** and **DataStructures\date\date.h** contains the *date class*.

These class is just used to store the date but more importantly to check for its validity and compare it between other dates.

```
class date
{
public:
    //Methods
private:
    string day;
    string month;
    string year;
};
```

Binary AVL Tree

The files **DataStructures\binaryAvlTree\tree.cpp** and **DataStructures\binaryAvlTree\tree.h** contains the *treeNode class*.

I store the citizens in a self balanced tree in order to have O(log(n)) time complexity for insetion, deletion and search. As for time complexity for InOrder , PreOrder and PostOrder traversal its O(n).

The key in the tree is the citizenID that is stored in the citizenRecord *citizen pointer. I access the id that way in order not to store an integer in the treeNode for less data duplication.

```
class treeNode
{
public:
    //Methods
private:
    treeNode *left;
    treeNode *right;
    citizenRecord *citizen;
    int balanceHeight;
};
```

Bloom Filter

The files **DataStructures\bloomFilter\bloomFilter.cpp** and **DataStructures\bloomFilter\bloomFilter.h** contains the *bloomFilter class* and *bloomFilterList class*.

The bloom filter class is just a char type array that uses bitwise shifting to manipulate the bits. I used char because sizeof(char) equals 1 Byte and it was helpful for calculations. For hashing i used the **djb2**, **sdbm** and **hash_i** provided.

I created a list of type bloomFilterList where i store a bloom filter for every new virus that i come across.

```
class bloomFilter
{
public:
    //Methods
private:
    int bloomSize;
    char *array;
};
```

```
class bloomFilterList
{
public:
    //Methods
private:
    int bloomSize;
    stringList *virus;
    bloomFilter *bloom;
    bloomFilterList *next;
};
```

SkipList

The files **DataStructures\skipList\skipList.cpp** and **DataStructures\skipList\skipList.h** contains the *skipList_List class* , *skipList class*, *skipListLevel class* and *skipListNode class*.

- skipList_List : Is a class that holds the 2 skip lists for every virus.
- skipList : Is the skipList. It holds the top and bottom level.
- skipListLevel : Holds informations about the level. Links the Levels together and points to the list of the level.
- skipListNode : Is the list that each level points to.

Every levels first node is minus infinity and last node is positive infinity.

```
class skipList_List
{
public:
    //Methods
private:
    stringList* virus;
    skipList* vaccinated;
    skipList* notVaccinated;
    skipList_List* next;
};
```

```
class skipList
{
public:
    //Methods
private:
    skipListLevel* ceiling;
    skipListLevel* floor;
};
};
```

```
class skipListLevel
{
public:
    //Methods
private:
    int myLevel;
    skipListNode* list;
    skipListNode* pos_inf;
    skipListLevel* downLevel;;
};
```

```
class skipListNode
{
public:
    //Methods
private:
    citizenRecord* citizen;
    skipListNode* next;
    skipListNode* down;
};
```

Population

The files **DataStructures\population\population.cpp** and **DataStructures\population\population.h** contains the *population class*.

This class is used in the /populationStatus & /popStatusByAge commands. Its purpose is to have all the counts stored and easy accesable throw a list.

Every population node is for one country.

- The inRange is the counts that are **in the range** of the dates given and are YES.
- The outRange is the counts that are **not in the range** of the dates given and are YES.
- The No is the counts that are **NO**. Here the dates **range doesnt matter** as NO doesn't have date.
- The noInformation is the counts that we **dont have information about that virus** but that person is in that country.

In order to have counts for the 4 age groups the inRange, outRange, No and noInformation are lists that holds a count. The first node is for the first age group the second node is for the seconds age group and so on.

If we are in the /populationStatus that has no age groups then the first node is general(for all the ages).

```
class population
{
public:
    //Methods
private:
    stringList *countryName; // key

    yes *inRange;
    yes *outRange;
    no *No;
    noneInfo *noInformation;

    population *next;
};
```

Commands

/vaccineStatusBloom

Must have exactly 2 arguments. At the start i check:

- id is only numbers
- 0 <= id <= 9999

If the virus is not in the virus list it is obvius we dont have to look the bloom filter to see that this id has not made the virus so return NOT **VACCINATED**. If we have that virus we hash the id and check the 16 bits to be set to 1. If all of them are 1 then **MAYBE** the id has been vaccinated fot that virus. If at least on bit is 0 then surely the id is **NOT VACCINATED**.

/vaccineStatus

Must have 1 or 2 arguments. At the start i check:

- id is only numbers
- 0 <= id <= 9999

If a virus name is given:

- Then if we dont have that virus in the virus list then return **NOT VACCINATED**.
- If the virus exists in the virus list, check in vaccinated skip list of that virus to see if this id exists. if yes return **VACCINATED ON dd-mm-yyy** if no return **NOT VACCINATED**.

If a virus name is **not** given do the above for every virus in the virus list.

/populationStatus

Must have 3 or 4 arguments. date1 < date2

The general idea is to make only one In Order search in the tree 0(n) and collect all the data that i need to create the statistics.

If a country is given:

- Create only one node of the population class for that country. This node will hold all the counts that i need.
- For every citizen:
  - if he comes from this country, look in his list status to find the virus.
    - if the virus was found with YES do +1 in the inRange count or the outRange count accordingly.
    - if the virus was found with NO do +1 in the No count.
    - if the virus was not found do +1 in the noInfo count.
  - if not from this country do nothing.
- The statistics are:

  > count = YES_in_range percentage = YES_in_range / (YES_in_range + YES_out_range + NO + none_information) * 100

If a country is **not** given:

- Create one node of the population class for every country. This list will hold all the counts that i need for every country.
- For every citizen get his country and **find** the population node of that country:
  - if this citizen has YES in the list status fot that virus do +1 in the inRange count or the outRange count accordingly of the found population node above.
  - if this citizen has NO in the list status fot that virus do +1 in the No count of the found population node above.
  - if this citizen has no information for that virus do +1 in the noInfo of the found population node above.

/popStatusByAge

Must have 3 or 4 arguments. date1 < date2

Its the same logic with /populationStatus. But we have to look for the age group that the citizen belongs to. There are 4 nodes for each count in order to implement the age groups. Check in the population class to see the logic.

- if he is in the age group 0-20 then its the first count
- if he is in the age group 20-40 then its the second count (->next)
- if he is in the age group 40-60 then its the first count (->next->next)
- if he is in the age group 60+ then its the first count (->next->next->next)

In order to access every age group we give an argument to the add that specifies the age group, that is the specific node.

> 0 for 0-20 1 for 20-40 2 for 40-60 3 for 60+

The statistics are:

- 0-20 :

  > count = YES_in_range_first percentage = YES_in_range_first / (YES_in_range_first + YES_out_range_first + NO_first + none_information_first) * 100

- 20-40 :

  > count = YES_in_range_second percentage = YES_in_range_second / (YES_in_range_second + YES_out_range_second + NO_second + none_information_second) * 100

- 40-60 :

  > count = YES_in_range_third percentage = YES_in_range_third / (YES_in_range_third + YES_out_range_third + NO_third + none_information_third) * 100

- 60+ :

  > count = YES_in_range_fourth percentage = YES_in_range_fourth / (YES_in_range_fourth + YES_out_range_fourth + NO_fourth + none_information_fourth) * 100

/insertCitizenRecord

This command behaves the same with inserting records from the inputFile.txt **BUT** if it is a duplicate with the same virus we will check the YES/NO. Cases:

- Already inserted citizen for that virus is **NO** and the citizen we try to insert has **YES** then we change the already inserted citizen from NO -> YES DATE. [print(status changed)]
- Already inserted citizen for that virus is **NO** and the citizen we try to insert has **NO** then we dont do anything. [print(already NO)]
- Already inserted citizen for that virus is **YES** and the citizen we try to insert has **NO** then we dont do anything. [print(has made the vaccine)]
- Already inserted citizen for that virus is **YES** and the citizen we try to insert has **YES** then we dont do anything. [print(has made the vaccine)]

/vaccinateNow

Just calls the command /insertCitizenRecord with the aguments given + YES + TODAY DATE cause the have the same functionality.

/list-nonVaccinated-Persons

Just prints the 0 level of the not vaccinate skiplist of that virus.

Script

The script is in the **script\testFile.sh**.

The script uses arrays to store the coutries and viruses. Also uses 2 additional arrays to store the ids that have been used before and the credentials of the records that have been already produced.

The is a function that creates a new record. That is used when creating a new record with unique id and store it to the 2 arrays.