Assignment #9

(due on Thursday, 11/18, 11:59 PM)

Please compress your SOURCE FILES (e.g., hw9_1_LastName _FirstName.cpp) into a zip file (hw9_LastName_FirstName.zip) and submit it to Blackboard. You should not submit project files or executable files; however, try to run your program successfully before you submit your source file. **You have to use proper indents and comment on your code. I also recommend you write pseudocode first. No late assignment!! Never show or talk about your code to your classmates.** If you have a problem submitting your file to Blackboard, you can email your assignment to Prof. Lee before the deadline.

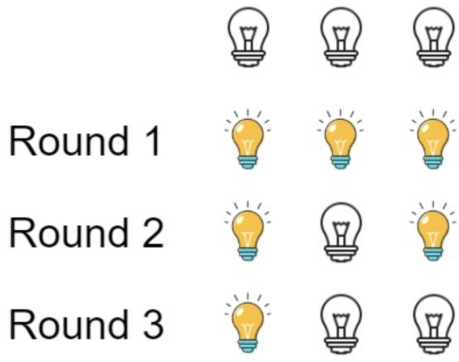
Grading:

Correctness	No errors, input/output correct, output presented nicely	60%
Solution	Code is efficient but easy to follow	20%
Style	Variable names, **comments**, indentation	20%

Q1. (50%)
There are **n** bulbs that are initially off. Write a C++ program that performs a series of toggling operations in a specified order. When toggling, one switches from one position to another (two total positions need to be specified). First, turn on all the bulbs, then turn off every second bulb, then every third bulb, and so on.

For example, in the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). **In the i^{th} round, toggle every i^{th} bulb. If the number of rounds is greater than the number of bulbs, you should wrap around to the start of the bulbs when toggling (first iteration).**

Additionally, return the number of bulbs that are on after the last round.
Below is an illustration of how the program would work for 3 bulbs over 3 iterations:




```
Input: n == 3, r == 3
Output: 1
Explanation: At first, the three bulbs are
[off, off, off].
After the first round, the three bulbs are
[on, on, on].
After the second round, the three bulbs are
[on, off, on].
After the third round, the three bulbs are
[on, off, off].
```

The number of bulbs (n) and the number of iterations (r) are to be taken as command-line arguments.
(*Continued on next page*)

Your outputs should be the same as the sample runs below. You can test different combinations by downloading the demo app (Windows, macOS Intel, macOS M1) from Blackboard.

Sample run #1: (No command-line arguments)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref
*** Bulb Switcher ***
Usage: ./hw9_1_ref NumberOfBulbs NumberOfIterations
```

Sample run #2: (Incorrect number of command-line arguments)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 5
*** Bulb Switcher ***
Usage: ./hw9_1_ref NumberOfBulbs NumberOfIterations
```

Sample run #3: (The command-line arguments are not numbers)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref Five Three
The number of bulbs has to be a positive integer.
The number of iterations has to be a positive integer.
```

Sample run #4: (The number of bulbs is not a positive number.)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 0 5
The number of bulbs has to be a positive integer.
```

Sample run #5: (The number of iterations is not a positive number.)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 3 0
The number of iterations has to be a positive integer.
```

Sample run #6: (Both the number of bulbs and the number of iterations are not positive numbers.)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref -100 -50
The number of bulbs has to be a positive integer.
The number of iterations has to be a positive integer.
```

Sample run #7: (n == 3, r == 3)(singular for bulb)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 3 3
Number of Bulbs: 3
Number of Iterations: 3
Round 0: [off, off, off]
Round 1: [on, on, on]
Round 2: [on, off, on]
Round 3: [on, off, off]
1 bulb is on.
```

Sample run #8: (n == 5, r == 3)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 5 3
Number of Bulbs: 5
Number of Iterations: 3
Round 0: [off, off, off, off, off]
Round 1: [on, on, on, on, on]
Round 2: [on, off, on, off, on]
Round 3: [on, off, off, off, on]
2 bulbs are on.
```

(*Continued on next page*)

Sample run #9: (n == 7, r == 4)

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 7 4
Number of Bulbs: 7
Number of Iterations: 4
Round 0: [off, off, off, off, off, off, off]
Round 1: [on, on, on, on, on, on, on]
Round 2: [on, off, on, off, on, off, on]
Round 3: [on, off, off, off, on, on, on]
Round 4: [on, off, off, on, on, on, on]
5 bulbs are on.
```

Sample run #10: (n == 4, r == 10)(The number of rounds is greater than the number of bulbs).

```
ming@Ming-Luns-Mac-mini-2 HW9 % ./hw9_1_ref 4 10
Number of Bulbs: 4
Number of Iterations: 10
Round 0: [off, off, off, off]
Round 1: [on, on, on, on]
Round 2: [on, off, on, off]
Round 3: [on, off, off, off]
Round 4: [on, off, off, on]
Round 5: [off, on, on, off]
Round 6: [off, off, on, on]
Round 7: [off, off, off, on]
Round 8: [off, off, off, off]
Round 9: [on, on, on, on]
Round 10: [on, off, on, off]
2 bulbs are on.
```

Requirements:

1. Your program must get the number of bulbs and the number of iterations from the command-line arguments.
2. Your program should define an object of the `std::vector<bool>` template class after it gets the number of iterations. You can use `true` for on and `false` for off.
3. You should use the `std::stoi` function to convert a command-line argument (a string) to an integer. You need to practice how to use `enum` for the `argv` index (e.g., `argv[FILENAME]`). You can refer to `std::stoi` via https://www.cplusplus.com/reference/string/stoi/

Hint:

You may use the following screenshot as the opening of your program.

```cpp
#include <iostream>
#include <cstdlib>  // for EXIT_FAILURE
#include <vector>   // for std::vector

enum {FILENAME, NUMOFBULBS, NUMOFITERATIONS, ARGCOUNT};  // 0, 1, 2, 3 for argv

int main(int argc, char const *argv[])
{
    int numOfBulbs;
    int numOfIterations;
    if(argc != ARGCOUNT){
        std::cout << "*** Bulb Switcher ***\n";
        std::cout << "Usage: " << argv[FILENAME] << " NumberOfBulbs NumberOfIterations\n";
        exit(EXIT_FAILURE);
    }
    // get number of bulbs using std::stoi
    // get number of iterations using std::stoi
    // input validation and error report
```

(*Continued on next page*)

Q2. (50%)

This assignment will be a little bit different than normal. You will be given a template file with the `main()` function already completed. Without changing the code in `main()`, you must make the program print out the following output:

```
List of People:
Cash Close
   Address: Off Campus
   Age: 21
Donald Duck
   Address: Disneyland
   Age: 90
Eric Cartman
   Address: South Park
   Age: 24
Homer Simpsons
   Address: Springfield
   Age: 32
Mickey Mouse
   Address: Disney World
   Age: 94
Minnie Mouse
   Address: Disneyland
   Age: 94
```

You should define a struct called `PERSON` and implement at least the following three functions:

- `createPerson()`: It takes the name, address, and age as arguments and returns a `PERSON` struct.
- `addPerson()`: It takes a reference to the `people` vector and a `PERSON` struct. It does not return anything. Use this function to add a person to the vector.
- `printPeople()`: It takes a reference to the `people` vector and a `PERSON` struct. It does not return anything. Use this function to print the entire list.

In the `main()` function, `createPerson()` gets called several times with different arguments. **DO NOT** make several `createPerson()` functions. Make one function with **default arguments**. You can infer the default arguments from the above outputs.

Your program should also sort the `people` vector by name alphabetically. You may do it in the `createPerson()` or `printPeople()` function.

Hint:
Refer to the following web pages for available `std::vector` and `std::string` member functions.

http://www.cplusplus.com/reference/vector/vector/
http://www.cplusplus.com/reference/string/string/