BME506
Fall 2023 – Lab 2
Language Basics – Arrays and Command-Line Arguments

Duration: one week.

Note:

1. Every lab assignment must be done individually.
2. When you name a folder or a file, you **should** avoid spaces in those names. For example, if you need to name a folder as **GreenApple**, you should name it as **GreenApple** instead of **Green Apple.** Naming a folder or a file without spaces in the names is to avoid compilation error that may occur while using MinGW build tools (For details, you may see here: http://www.mingw.org/wiki/Getting_Started).
3. **ALL code should be compiled and run on the laboratory computers before submission.**

Goals:

In the following lab, you will continue to work with refining your command of core C/C++ programming constructs, paying particular attention to the use of **1D** and **2D arrays**, **use of command-line arguments** and **use some aspects of `cout` formatting**.

- Please indent your code properly, include appropriate comments, and use proper names for functions and variables.
- The prototypes of the functions should be in a header file named **Lab2.h.**
- The implementation of the functions should be in a file named **Lab2.cpp.**
- The function **main** should also be in **Lab2.cpp.**
- The code of **Lab2.h** and the partial implementation of **main** (that should go in **Lab2.cpp**) has already been provided in the **Appendix** at the end of this file. You **must** use them in your implementation.

General steps to build the C++ project in Netbeans for Lab2:

1. Create a Netbeans project named **Lab2Proj** while taking care of the NOTE below.
   NOTE: While creating the project, please **make sure**
   - you set the field "**Project Location:**" appropriately. **Be sure to remember this project location** path. (To keep track of your **Project Location** you may do as follows: Once you have created the project, open a File Explorer, and go into your **Project Location** folder).
   - once you have chosen your **Project Location**, enter the word **Lab2Proj** in the field "**Project Name:**". Notice that your project name gets appended to your **Project Location** path.
   - you **UN-CHECK** the checkbox on the left of "**Create Main File**" (We do so since we do not need any file named main.cpp file in this project).
   - you Click **[Finish]** to create the project.

- a new folder named **`Lab2Proj`** has got created automatically under your **Project Location** folder. This **`Lab2Proj`** folder is your Netbeans ***project folder***.

2. Under netbeans project **Lab2Proj**, right-click on the "Header Files" > "New" > "C++ Header File …". Set the field **"File Name:"** as **`Lab2`**. Set the **"Extension:"** as "**`h`**". Click **[Finish]**.
3. The file **`Lab2.h`** is created. Copy the content for the header file from Appendix into this newly create file.
4. Under netbeans project **Lab2Proj**, right-click on the "Source Files" > "New" > "C++ Source File …". Set the field **"File Name:"** as **`Lab2`**. Set the **"Extension:"** as "**`cpp`**". Click **[Finish]**.
5. The file **`Lab2.cpp`** is created. Copy the content for the source file from Appendix into this newly create file.
6. In **Lab2.cpp**, add the relevant functions that are intended to solve problems of **Part I**, **Part II**, and **Part III** given below.
7. Note: At this point, make sure that your two newly created files **`Lab2.h`** and **`Lab2.cpp`** exists in the **`Lab2Proj`** folder.
8. Right-Click on project **Lab2Proj** >
   Select "Properties" >
   Click "Run" >
   On right, choose "Console Type" to be "Standard Output"
   Click [Apply]
   Click [OK]
9. Build the **Lab2Proj** project.
10. Open a Command Prompt window. Go to the folder **`Lab2Proj\dist\Debug\MinGW-Windows\`** (where **`Lab2Proj`** is the Netbeans *project folder*). You should see that the executable **lab2proj.exe** has got created in this folder.
11. Run the executable **lab2proj.exe** from the command prompt.

Important Notes:

1. When your code is error-free, it should build successfully (as seen at the compilation process at Netbean's "Output" window). **However,** at times, in the Netbean's *source editor* window, you may notice red line below a built-in method, such as `size()` method of built-in type string. You may notice this even if your program has built successfully—For example, in the invocation `str1.size()`, you may see a red line below the word `size`. This is misleading. Such a red line should disappear if you perform the following steps:

   Right-Click on project **Lab2Proj** >
   Select "Code Assistance" >
   Select "Clean C/C++ cache and restart IDE"

2. **If you ever want to remove a project from Netbeans,** right-click on project > click **"Close". AVOID** using **"Delete"** to remove a project from the Netbeans. If you ever remove a project using "Delete", you will have to create a new project all over again to replace the "deleted" project.

In this lab, we will implement a series of functions that:

- simulate an exponential growth equation to model and log bacterial growth.
- allow to model the growth of multiple bacteria (different populations), store results in a 2D array and display the growth of populations in a formatted table

You will need to implement 4 functions:

- **initialize** – it initializes the bacterial sample (prompt the user for *k* and N_0).
- **calculate** – it calculates and store the bacterial population (over time t=0 to 10 hours) in an array.
- **display** – it outputs the growth summary of the single bacterial population from the 1D array, on console
- **twoDdisplay** - it outputs the growth summary of multiple bacterial populations from the 2D array, on console

PART I: Simulate exponential bacterial cell growth

1. This part allows the user to enter parameters *k* and N_0 that initialize a simple population of bacterial cells *N*(*t*), which grows according to the exponential growth equation:

   $\mathbf{N(t) = N_0\, e^{kt}}$; *where t=time (hours),* N_0 *= initial population (cells); k = growth factor*

   Examples of possible output from this part:

```
Initializing Bacteria:
        Growth factor (k) [0.0-1.0] : 0.13
        Initial population (N0) [1-1000] : 25

        Growth Summary:

        Hour    Population
        ====    ==========
        0       25.000
        1       28.471
        2       32.423
        3       36.925
        4       42.051
        5       47.889
        6       54.537
        7       62.108
        8       70.730
        9       80.550
        10      91.732
```

```
Initializing Bacteria:
        Growth factor (k) [0.0-1.0] : 0.34
        Initial population (N0) [1-1000] : 112

        Growth Summary:

        Hour    Population
        ====    ==========
        0       112.000
        1       157.354
        2       221.074
        3       310.598
        4       436.374
        5       613.082
        6       861.348
        7       1210.149
        8       1700.196
        9       2388.686
        10      3355.979
```

2. Compile the project **Lab2Proj**. Open a Command Prompt window. Go to the folder where the executable **lab2proj.exe** is located. Run the executable from the command prompt (>) as follows:

> **lab2proj.exe**

PART II: Pass the values of *k* and N_0 to the program as command-line arguments

1. In this case, we will use $k = 0.24$ and $N_0 = 38$ as example values. When k and N_0 are specified at command-line, the user should **not** be prompted to specify k and N_0. The output should reflect as shown below.

```
Bacteria Initialized:
        Growth factor (k) = 0.24
        Initial poulation (N0) = 38

        Growth Summary:

        Hour    Population
        ====    ==========
        0       38.000
        1       48.307
        2       61.411
        3       78.068
        4       99.244
        5       126.164
        6       160.386
        7       203.891
        8       259.196
        9       329.503
        10      418.881
```

3. Compile the project **Lab2Proj**. Open a Command Prompt window. Go to the folder where the executable **lab2proj.exe** is located. Run the executable at the command prompt (>) where k and N_0 are specified. For example, when k is specified as 0.24 and N_0 is specified as 38, the command-line should look like:
   > **lab2proj.exe 0.24 38**

PART III: Pass the number of bacterial populations as a command-line argument

1. The following example command-line implies that 3 separate bacterial populations are to be tracked and logged when the program is run at the command prompt (>):
   > lab2proj.exe 3

   An example output is shown below:

```
Initializing Bacteria:
        Growth factor (k) [0.0-1.0] : 0.2
        Initial population (N0) [1-1000] : 20

Initializing Bacteria:
        Growth factor (k) [0.0-1.0] : 0.11
        Initial population (N0) [1-1000] : 80

Initializing Bacteria:
        Growth factor (k) [0.0-1.0] : 0.35
        Initial population (N0) [1-1000] : 12

        Agar Summary:

         Hour    Population (0)    Population (1)    Population (2)
        ============================================================
            0            20.000            80.000            12.000
            1            24.428            89.302            17.029
            2            29.836            99.686            24.165
            3            36.442           111.277            34.292
            4            44.511           124.217            48.662
            5            54.366           138.660            69.055
            6            66.402           154.783            97.994
            7            81.104           172.781           139.060
            8            99.061           192.872           197.336
            9           120.993           215.299           280.033
           10           147.781           240.333           397.385
```

2. Compile the project **Lab2Proj**. Open a Command Prompt window. Go to the folder where the executable **lab2proj.exe** is located. Run the executable at the command prompt (>) where the number of separate bacterial populations is specified. An example command-line where 3 separate bacterial populations is specified would be as follows:
   > lab2proj.exe 3

Lab Submission

Deadline: See announcement in D2L for deadline.

ALL code should be compiled and run on the laboratory computers before submission.

Create a folder. Name it as YourLastname_YourFirstname_bme506_Labnumber.
Example: If student name is John Smith, the name of folder should be Smith_John_bme506_Lab2.

Copy your Netbeans ***project folder***, **Lab2Proj**, in the above folder. Make sure the folder **Lab2Proj** contains your two files `Lab2.h` and `Lab2.cpp`.

The above folder must also contain a duly filled and signed standard cover page. The cover page can be found on the departmental web site: Standard Assignment/Lab Cover Page

Compress the above folder as a single zip file that is named according to the following rule:
YourLastname_YourFirstname_bme506_Labnumber.zip.
Example: Smith_John_bme506_Lab2.zip.

Upload the above zip file on D2L through the "Assessment" > "Assignments" link.

Note: If the code does not compile, the submission will receive a ZERO mark.

Appendix

#include <iomanip>
is a library with functions that can manipulate the formatting of variables passed to the output stream (cout). For instance, using the function '`setw(X)`' in an output stream will use up to X characters to display the next variable in the stream (right-justifying the output).

```
cout  <<  setw(8) << 2.6796 << endl;
cout  <<  setw(8) << 2.67 << endl;
---------------------------------
  2.6796
    2.67
```

Similarly, functions like `setprecision(X)` can be used within the stream to control the number of significant digits, or decimal places to display when outputting variables.

[*see http://www.cplusplus.com/reference/iomanip/ for more details and examples of usage: particularly the links relating to setw, setprecision, setfill and setbase (which is used to convert output to hexadecimal, binary and formats relating to bases other than decimal)*]

Command line arguments
We can pass arguments directly into our program at runtime (without having to request input from *cin*). For instance, if we call the executable of our program with an extended string that includes a whitespace separated list of string values, then these may be accessible by the main() routine. To enable such usage, main() must be declared in order to accept these arguments:

```
int main(int argc, char *argv[]) { … }
```

Here, *argc* contains the number of whitespace separated strings on the command-line, and *argv* is an array of pointers to char objects (i.e. null terminated strings). For example, if our executable was called "myexe", and we called the program from the terminal using:

```
myexe hello 4 3.812 bme506
```

The command line string contains 5 whitespace separated substrings (*argc* = 5); and each would be stored as a string in the *argv* array:

argv[0] = “myexe”
argv[1] = “hello”
argv[2] = “4”
argv[3] = “3.812”
argv[4] = “bme506”

We can therefore extract and use these arguments to control our program (e.g. as input to control flow statements like if/then/else or switch).

#include <cstdlib> provides access to standard C library functions (e.g. *atoi*( ) or *atof*( )) which can be used to convert strings of numeric values into numeric values (when assigning to int’s or double’s)

e.g. `int myInt = atoi(argv[4]);`

In your Lab2.h file, the code should be as follows:

```
#ifndef LAB2_H_
#define LAB2_H_
#include <iostream>

using namespace std;

void initialize(double& k, int& n0);
void calculate( const double k,  const int n0, double array[]);
void display(const double array[]);
void twoDdisplay(int num, const double arr[][11]);

#endif /* LAB2_H_ */
```

In your Lab2.cpp file, the #include statement and the function `main` should be as follows. Fill the body of function `main` as required.

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include "Lab2.h"


//implement the function initialize
//implement the function calculate
//implement the function display
```

```
//implement the function twoDdisplay


int main(int argc, char* argv[]) {
      double k;
      int n0;
        if (1 == argc) {  //Part I
            //WRITE THE CODE FOR Part I HERE.
            ...
        }
        else if (2 == argc) {  //Part III
            //WRITE THE CODE FOR Part III HERE.
            ...

        }
        else if (3 == argc) {  //Part II
            //WRITE THE CODE FOR Part II HERE.
            ...

        }
        else {
            cout << "Number of command-line arguments "
                 << "(including the name of the program) "
                 << "should be three or less." << endl;
        }
} //end of main
```