CS 161 Lab #4 –1-D Arrays and Functions

- Start by getting checked off for (up to) 3 points of work from the previous lab in the first 10 minutes.
- Your goal is to get all points for Lab 4 checked off by a TA by the end of this lab, so that you do not need to do additional work outside of the lab.

Goals:

- Practice declaring, accessing, and updating arrays
- Practice writing and using functions

(5 pts) A. Working with 1-dimensional Arrays

Your TAs will introduce examples of **using 1-D arrays**. Pay close attention and ask questions about any details that are confusing.

Next, **you will work with other student(s) in a small group.** Go through the steps together and help each other figure things out! You should each complete all steps. Ask for a TA if you are stuck.

(3 pts) Tracking fruit fly populations

Working with your partners, design a program to allow a scientist to enter the number of fruit flies in each of 5 jars. After the data has been entered, the program should print out the population of each jar.

Implementation:

- Declare an array of 5 variables to hold the fruit fly counts for each jar. Choose an appropriate data type.
- Read in each user input inside a loop and store the input in the array.
- Use another loop to print out the population of each jar.

After writing the program **(lab4_fruitfly.cpp)**, run your test cases and check that they work correctly. You do not need to check user input for validity (you will be the user ☺).

Example output (user input is highlighted):

```
Hello Scientists! You have 5 jars to fill with fruit flies.
How many fruit flies would you like in jar 0: 3
How many fruit flies would you like in jar 1: 2
How many fruit flies would you like in jar 2: 5
How many fruit flies would you like in jar 3: 1
How many fruit flies would you like in jar 4: 4
Number of fruit flies per jar:
Jar 0: 3
Jar 1: 2
Jar 2: 5
Jar 3: 1
Jar 4: 4
```

(2 pts) Analyze fruit fly populations

Working with your partners, decide how to update your program to report the index of the jar with the largest population and the index of the jar with the smallest population.

- Add another loop to find the largest and smallest values in the array.
    - Use a new variable (declared before the loop) to store the index of the largest jar and another variable to store the index of the smallest jar.
    - Inside the loop, check each jar's contents and update the largest/smallest index if appropriate.
- Print out the index of the jar with the largest and the jar with the smallest population.

Before writing the program, create **3 test cases** (all 5 user inputs (fruit fly counts per jars), plus with expected output). Put your test cases in comments in the program.

Example output for the inputs given above:

```
The jar with the most fruit flies: jar 2 (5 fruit flies)
The jar with the least fruit flies: jar 3 (1 fruit flies)
```

If you have extra time, update the program to use "fly" (singular) if there is only one fruit fly:

```
The jar with the most fruit flies: jar 2 (5 fruit flies)
The jar with the least fruit flies: jar 3 (1 fruit fly)
```

(2 pts) B. Design Functions to Operate on Integers

Your TAs will introduce examples of **writing functions**. Pay close attention and ask questions about any details that are confusing.

Working with your partners, develop a mini-design for each of the following integer operations. Your mini-design for each one will describe the operation of a function. For each one, make a flowchart and at least 5 test cases.

- **bool is_even(int num): return true if num is even and otherwise return false.** Recall that you can check if x is even with `if (x%2 == 0)`
    - What are your 5 test cases? (Don't forget negative numbers!)
- **int product(int num1, int num2): returns the product of multiplying two numbers**. For example, if num1 is 3 and num2 is -7, the function should return -21.
    - What are your 5 test cases?

When you have finished your designs, ask for a TA to come to your breakout room and check your work. You can describe your design verbally or share your screen to show it.

(3 pts) C. Implement Functions to Operate on Integers

Once your designs are checked off by a TA, proceed to implementation.

1. Create a file named `lab4_functions.cpp` .
2. Create a `main()` method that will call your functions using each of the test cases from your design.
3. Implement the `is_even()` and `product()` functions **above** `main()` in the file. Include the function headers that are defined for you below.
4. Put your test cases in the function header: input and expected return value.

5. Compile and run your program. Check your test cases and refine your program until all of them pass.

```
/*************************************************************
 ** Function: is_even
 ** Description: return true if num is even
 ** Parameters: an integer, num
 ** Test case 1: input _____, return _____
 ** (More test cases here)
 *************************************************************/
bool is_even(int num)
{
  ...
}

/*************************************************************
 ** Function: product
 ** Description: return the product of two integers
 ** Parameters: two integers: num1 and num2
 ** Test case 1: inputs _____ and _____, return _____
 ** (More test cases here)
 *************************************************************/
int product(int num1, int num2)
{
  ...
}
```

D. Get your work checked off by a TA.

Submit your programs on Canvas (this can be done while you are waiting to be checked off).
All partners' names must be in the comment header to get credit.

1. Transfer your .cpp files from the ENGR servers to your local laptop.
2. Go to the Lab 4 assignment: https://canvas.oregonstate.edu/courses/1770357/assignments/7847925
3. Click "Submit Assignment".
4. Upload your .cpp files (lab4_fruitfly.cpp, lab4_functions.cpp).
5. Click the **"Submit Assignment"** button.
6. You are done!

If you finish the lab early, this is a golden chance to work on your Assignment 3 design (with TAs nearby to answer questions!).

Remember, the design you submit must be your work alone (no partners).
