Project #4

Due Oct 4 by 11:59pm **Points** 100 **Available** until Oct 5 at 2am

Task #1 - Using pointers to process arrays (50 points)

Task

Modify project 3, task #2, to use pointers to process the array in the active_seconds function.

Requirements

- Name your program **project4_escalator.c**
- The program should include the following function:

  int active_seconds(int arrival[], int n)

  The function calcuates the total number of active seconds. Array arrival[] represents the arrival time of each person, n is the length of array arrival, representing the number of people.

  This function should use pointer arithmetic– not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

- In the main function, call the **active_seconds** function and display the result.
- Pointer arithmetic is NOT required in the main function.

Examples (your program must follow this format precisely)

Example #1

```
Enter number of people: 2
Enter arrival times: 12 25
Active seconds: 20
```

Example #2

```
Enter number of people: 3
Enter arrival times: 12 15 20
Active seconds: 18
```

Example #3

```
Enter number of people: 5
Enter arrival times: 12 15 20 34 41
Active seconds: 35
```

Submission instructions

1. Develop and test your program on the student cluster
   1. To compile your program, run: **$ gcc -std=c99 -Wall your_program.c**
   2. To execute your program, run: **$ ./a.out**
2. Name your program **project4_escalator.c**
   1. To rename your program, run: **$ mv your_program.c project4_escalator.c**
3. Test your program with the shell script on Unix: try_project4_escalator
   1. Upload the script to the same directory as your program.
   2. Run: **$ chmod +x try_project4_escalator**
   3. Run: **$ ./try_project4_escalator**
4. Download the program from student cluster and submit it on **Canvas->Gradescope**. Make sure you submit a file with the correct name!
5. You can submit your program as many times as needed before the due date. Gradescope will indicate the test cases with incorrect output, if any exists.

Task #2 - Exclusive or (50 points)

Task

The exclusive or of two arrays a and b of integers are the elements that are either in a or in b, but not in both a and b. Write a program that finds the exclusive or of two input arrays. For example, array a contains elements *{8, 10, 12}*, array b contains elements *{23, 4, 8, 12}*. The exclusive or of a and b should contain {10, 23, 4}.

Requirements

- Name your program **project4_exclusive_or.c**.
- Follow the format of the examples below.
- Your program should include the following function:

void exclusive_or(int *a, int n1, int *b, int n2, int *c, int *size);

The function should use pointer arithmetic – not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

The *exclusive_or* function finds the exclusive or of array a and array b, and stores the result in array c. n1 is the number of elements of the array a and n2 is the number of elements in array b. The function should keep track of the number of elements for the exclusive or and store it through the pointer variable size.

In the main function, ask the user to enter the lengths of the arrays, the array elements, declare the input arrays, and the output array. The length of the output array should be declared as the sum of the lengths of the two input arrays. The actual length will be determined by the exclusive_or function.

- **The main function call the exclusive_or function and display the result.**
- Pointer arithmetic is NOT required in the main function.
- The order of the numbers in the output does not matter.

Examples (your program must follow this format precisely)

Example #1

```
Enter length of array #1: 5
Enter array elements: 31 17 32 2 88
Enter length of array #2: 3
Enter array elements: 17 2 31
Output: 32 88
```

Example #2

```
Enter length of array #1: 3
Enter array elements: 36 19 32
Enter length of array #2: 3
Enter array elements: 17 2 31
Output: 36 19 32 17 2 31
```

Example #3

```
Enter length of array #1: 3
Enter array elements: 31 17 2
Enter length of array #2: 3
Enter array elements: 17 2 31
Output:
```

Submission instructions

1. Develop and test your program on the student cluster
   A. To compile your program, run: **$ gcc -std=c99 -Wall your_program.c**
   B. To execute your program, run: **$ ./a.out**
2. Name your program **project4_exclusive_or.c**
   A. To rename your program, run: **$ mv your_program.c project4_exclusive_or.c**
3. Test your program with the shell script on Unix: try_project4_exclusive_or
   A. Upload the script to the same directory as your program.
   B. Run: **$ chmod +x try_project4_exclusive_or**
   C. Run: **$ ./try_project4_exclusive_or**
4. Download the program from student cluster and submit it on **Canvas->Gradescope**. Make sure you submit a file with the correct name!
5. You can submit your program as many times as needed before the due date. Gradescope will indicate the test cases with incorrect output, if any exists.

General instructions

Grading

Task #1

Total points: 50

- A program that does not compile will result in a zero.
- Runtime error and compilation warning 3 points

  1 points off, if a warning is present.

  3 points off, if multiple warnings are present.

- Commenting and style 8 points

  1 point off for not putting name and description at the beginning

  2 to 4 points off if the code didn't have clarifying comments.

  1 to 3 points off (depending on how much indentation is off) if the program is not indented properly.

- Functionality

  3 points off for each incorrect test case

  25 points off if pointer arithmetic is not used to process array

  10-20 points off if pointer arithmetic is not used correctly to process array

Task #2

Total points: 50

- A program that does not compile will result in a zero.
- Runtime error and compilation warning 3 points

  1 points off, if a warning is present.

  3 points off, if multiple warnings are present.

- Commenting and style 8 points

  1 point off for not putting name and description at the beginning

  2 to 4 points off if the code didn't have clarifying comments.

  1 to 3 points off (depending on how much indentation is off) if the program is not indented properly.

- Functionality

  5 points off for each incorrect test case

  15 points off if pointer arithmetic is not used to process arrays

  5-10 points off if pointer arithmetic is not used correctly to process arrays

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

- Your program should begin with a comment that briefly summarizes what it does. This comment should also include your name.
- In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only needed in order for a reader to understand what is happening.
- Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
- Use consistent indentation to emphasize block structure.
- Full line comments inside function bodies should conform to the indentation of the code where they appear.
- Macro definitions (**#define**) should be used for defining symbolic names for numeric constants. For example: **#define PI 3.141592**
- Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
- Use underscores to make compound names easier to read: **tot_vol** and **total_volumn** are clearer than **totalvolumn**.