Name: _____

Lab for ITSC 3181, Introduction to Computer Architecture, Spring 2023

Lab #02: Basic C programming with pointer, array and memory, and number system. Due on Wednesday 1/25 and total points: 2% of the final accumulated percentage grade.

Programming exercise for pointers, memory address of variables and array elements, pointer arithmetic and referencing addresses using pointers. (80 points)

In this lab, you are given an incomplete C program `lab02_address_1Dstencil.c`, which can be downloaded from https://passlab.github.io/ITSC3181/notes/Lab_02/lab02_address_1Dstencil.c. Your work is to complete the program so it prints the output shown in the screenshot. There are 5 places that you need to add code to complete the program, identified as TODO #1 to TODO #5 in the code. Each TODO is assigned a certain amount of points you will earn if you complete it correctly. **For submision, you need to submit your completed code, screenshot that shows the command and output of running the program. The output should be exactly the same as the screenshot provided, though the memory address numbers could be different.**

The `lab02_address_1Dstencil.c` file for you to start with:

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]) {
    int a1 = 10;
    int a2 = 10;
    double b1 = 3.14;
    double b2 = 9.8;
    short c1 = 100;
    short c2 = 20;
    char d1 = 'a';
    char d2 = 'c';

    int M = 4;
    int N = 6;
    int A[M][N];
    int i, j;

    //You can find the memory address of a function by using & operator
    printf("------------------- The address of main function: %p ----------------------------\n", &main);

    printf("--- Variable memory addresses using & operator and variable sizes using sizeof operator ---\n");
    /** TODO #1 (10 points): add your code for printing addresses and sizes for variables a1, a2, b1, b2, c1, c2, d1, d2 */



    printf("\n");
    printf("--- Memory addresses of array elements using & operator and base+offset calcualtion ----\n");
    /** TODO #2 (20 points): add your code for printing addresses of array elements using & operator and base+offset calcualtion */



    /* 1-D stencil operation: for an array B[M], update each element by B2[i] = (B[i-1]+B[i]+B[i+1])/3 */
    srand(1<<12);   // Initialize random number generator seed, should only be called once.
    M = 100;
    int B[M];
    int *iterator;
    //generate rand number for array B and print array B
    printf("\n------------------------- 1-D stencil operation ---------------------------------------\n");
    printf("Element values of array B[%d]\n", M);
    for (i=0; i<M; i++) {
        //TODO #3 (10 points): update the iterator to store the address of element i of B.

        *iterator = rand() % 20; /* assign the array element a random value between 0 and 20 */
        printf("%d\t", *iterator);
        if ((i+1)%10==0) printf("\n"); //go to the next line
    }

    iterator = B;
    int B2[M];
    for (i=1; i<M-1; i++) {
        /* TODO #4 (35 points): perform operation B2[i] = (B[i-1]+B[i]+B[i+1])/3. You are only allowed to use
         * the iterator and i variable to calcualte the memory addresses of needed elements of B and B2.
         * You should NOT use [] or & operator for any purpose here */



    }

    /* boundary copy */
    *B2 = *B;
    *(B2+M-1) = *(B+M-1);

    printf("\nElement values of array B2[%d] after 1-D stencil operation on array B\n", M);
    for (i=0; i<M; i++) {
        //TODO #5 (5 points): update the iterator to store the address of element i of B2.



        printf("%d\t", *iterator);
        if ((i+1)%10==0) printf("\n"); //go to the next line
    }

    return 0;
}
```

The screenshot below shows the correct output of the program if it is completed:

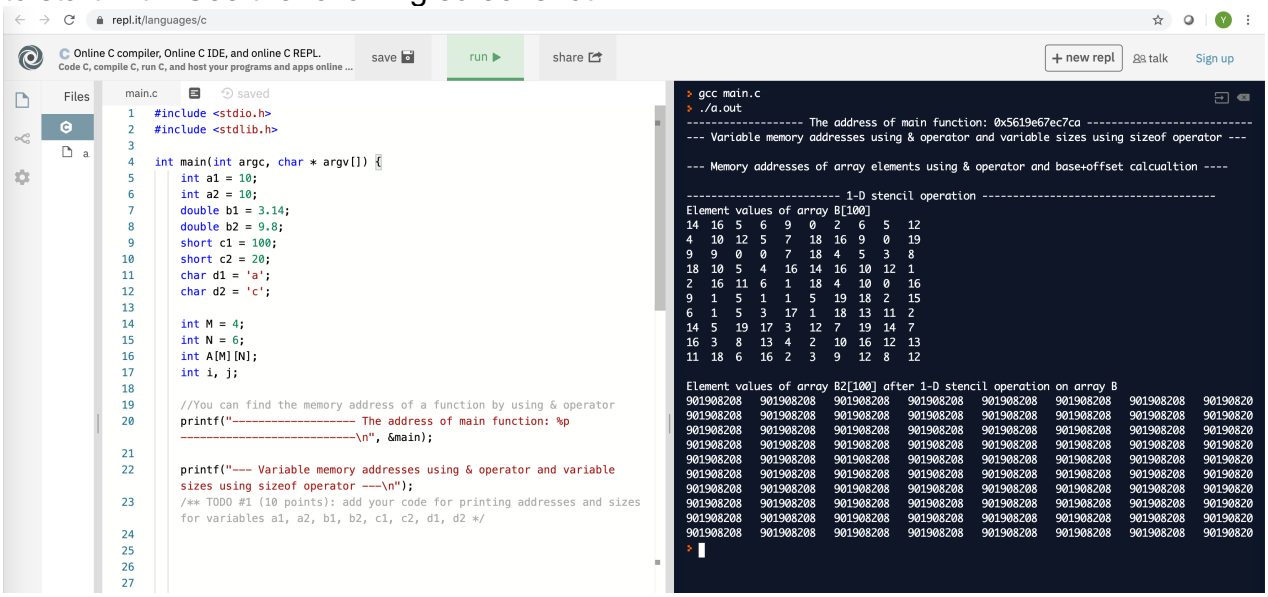
```
yanyh@vm:~$ ./a.out
------------------- The address of main function: 0x4006a6 ---------------------------
--- Variable memory addresses using & operator and variable sizes using sizeof operator ---
Memory address of variable a1: 0x7ffce61c718c, size: 4 bytes
Memory address of variable a2: 0x7ffce61c7190, size: 4 bytes
Memory address of variable b1: 0x7ffce61c71a8, size: 8 bytes
Memory address of variable b2: 0x7ffce61c71b0, size: 8 bytes
Memory address of variable c1: 0x7ffce61c7188, size: 2 bytes
Memory address of variable c2: 0x7ffce61c718a, size: 2 bytes
Memory address of variable d1: 0x7ffce61c7186, size: 1 bytes
Memory address of variable d2: 0x7ffce61c7187, size: 1 bytes

--- Memory addresses of array elements using & operator and base+offset calcualtion ----
Base memory address of array A[4][6]: 0x7ffce61c70b0
Memory address (&A[0][0]): 0x7ffce61c70b0, offset: 0000, base+offset: 0x7ffce61c70b0
Memory address (&A[0][1]): 0x7ffce61c70b4, offset: 0004, base+offset: 0x7ffce61c70b4
Memory address (&A[0][2]): 0x7ffce61c70b8, offset: 0008, base+offset: 0x7ffce61c70b8
Memory address (&A[0][3]): 0x7ffce61c70bc, offset: 000c, base+offset: 0x7ffce61c70bc
Memory address (&A[0][4]): 0x7ffce61c70c0, offset: 0010, base+offset: 0x7ffce61c70c0
Memory address (&A[0][5]): 0x7ffce61c70c4, offset: 0014, base+offset: 0x7ffce61c70c4
Memory address (&A[1][0]): 0x7ffce61c70c8, offset: 0018, base+offset: 0x7ffce61c70c8
Memory address (&A[1][1]): 0x7ffce61c70cc, offset: 001c, base+offset: 0x7ffce61c70cc
Memory address (&A[1][2]): 0x7ffce61c70d0, offset: 0020, base+offset: 0x7ffce61c70d0
Memory address (&A[1][3]): 0x7ffce61c70d4, offset: 0024, base+offset: 0x7ffce61c70d4
Memory address (&A[1][4]): 0x7ffce61c70d8, offset: 0028, base+offset: 0x7ffce61c70d8
Memory address (&A[1][5]): 0x7ffce61c70dc, offset: 002c, base+offset: 0x7ffce61c70dc
Memory address (&A[2][0]): 0x7ffce61c70e0, offset: 0030, base+offset: 0x7ffce61c70e0
Memory address (&A[2][1]): 0x7ffce61c70e4, offset: 0034, base+offset: 0x7ffce61c70e4
Memory address (&A[2][2]): 0x7ffce61c70e8, offset: 0038, base+offset: 0x7ffce61c70e8
Memory address (&A[2][3]): 0x7ffce61c70ec, offset: 003c, base+offset: 0x7ffce61c70ec
Memory address (&A[2][4]): 0x7ffce61c70f0, offset: 0040, base+offset: 0x7ffce61c70f0
Memory address (&A[2][5]): 0x7ffce61c70f4, offset: 0044, base+offset: 0x7ffce61c70f4
Memory address (&A[3][0]): 0x7ffce61c70f8, offset: 0048, base+offset: 0x7ffce61c70f8
Memory address (&A[3][1]): 0x7ffce61c70fc, offset: 004c, base+offset: 0x7ffce61c70fc
Memory address (&A[3][2]): 0x7ffce61c7100, offset: 0050, base+offset: 0x7ffce61c7100
Memory address (&A[3][3]): 0x7ffce61c7104, offset: 0054, base+offset: 0x7ffce61c7104
Memory address (&A[3][4]): 0x7ffce61c7108, offset: 0058, base+offset: 0x7ffce61c7108
Memory address (&A[3][5]): 0x7ffce61c710c, offset: 005c, base+offset: 0x7ffce61c710c

------------------------- 1-D stencil operation -------------------------------------
Element values of array B[100]
14      16      5       6       9       0       2       6       5       12
4       10      12      5       7       18      16      9       0       19
9       9       0       0       7       18      4       5       3       8
18      10      5       4       16      14      16      10      12      1
2       16      11      6       1       18      4       10      0       16
9       1       5       1       1       5       19      18      2       15
6       1       5       3       17      1       18      13      11      2
14      5       19      17      3       12      7       19      14      7
16      3       8       13      4       2       10      16      12      13
11      18      6       16      2       3       9       12      8       12

Element values of array B2[100] after 1-D stencil operation on array B
14      11      9       6       5       3       2       4       7       7
8       8       9       8       10      13      14      8       9       9
12      6       3       2       8       9       9       4       5       9
12      11      6       8       11      15      13      12      7       5
6       9       11      6       8       7       10      4       8       8
8       5       2       2       2       8       14      13      11      7
7       4       3       8       7       12      10      14      8       9
7       12      13      13      10      7       12      13      13      12
8       9       8       8       6       5       9       12      13      12
14      11      13      8       7       4       8       9       10      12
yanyh@vm:~$
```

1. Review the following information about C printf and memory address of variables and array elements.
    a. printf: http://www.cplusplus.com/reference/cstdio/printf/
    b. Pointers, array, address of array elements: lecture slides for C Basics (https://passlab.github.io/ITSC3181/notes/lecture02_CBasics.pdf)
2. Use https://repl.it/languages/c website to develop, compile and run your program. First, open the file from the web browser https://passlab.github.io/ITSC3181/notes/Lab_02/lab02_address_1Dstencil.c, go to https://repl.it/languages/c site, and cut-paste the content of the `lab02_address_1Dstencil.c` file to the editor. It is named as main.c file in website editor. Please ignore the file name. Then compile and run the program first (`gcc main.c; ./a.out`) to make sure you have the correct file to start with. See the following screenshot:



3. Edit the file and make changes for those TODOs. Each TODO are given with detailed comments for the task. The first two TODOs are for showing variable memory addresses, variable sizes and array element addresses. For TODO #1 (10 points), you will need add printf statements to print out the memory address (obtained using & operator) and the size (using sizeof operator) of each variable. For TODO #2 (20 points), you will need to add printf to print the base memory address of the array A. After that, you will need to add a double-nested loop. In each inner loop iteration, add printf to print the memory address of each array element obtained using &, the offset in bytes calculated using the formula we discussed in the class, and the sum of base (cast to char* type first) and offset. For each iteration, base+offset should be same as the address of the element you obtain using &. After this exercise, you should know that array references, e.g. A[3][2], A[i][j], B[i] are all address calculation operations, i.e. to calculate the address of a specific array element based on the formula of base+offset.
4. The last three TODOs (TODO #3, #4 and #5) are for implementing a simple 1-D stencil operation (a simplified version of the algorithm that is used for lots of image processing algorithms such as image blurring, enhancing, etc). The 1-D stencil operation is to update each array element of B[M] and store the results in another array B2[M] using the operation B2[i] = (B[i-1] + B[i] + B[i+1])/3 (i=1, 2, …, M-2). For the first and last element of B2 (boundary condition), we do B2[0] = B[0], and B2[M-1] = B[M-1]. You need to implement this operation using only pointers, the "`int *iterator`" variable, and array element access should use the address calculated based on the pointer arithmetic and array element address calculation (base+offset). TODO #3 (10 points) is for calculating the address of array element B[i] so value can be assigned to it. TODO #4 (35 points)

is for the operation B2[i] = (B[i-1] + B[i] + B[i+1])/3. In this TODO, your code should reference each needed array elements of B and B2 using pointers. Using & and [ ] operators to do it are not counted as your implementation. TODO #5 (5 points) is for calculating the address of array element B2[i] so values can be read for printing. ***It is important to note that when calculating address of an element of an array, you calculate the byte address (casting to char *), but when you assign that byte address to an "int * ptr" variable, or use that pointer as address to read or write a value from/to that address (by using * operator), you will need to specify the type of the pointer (e.g. int * by casting). E.g.*** `int * ptr; ptr = (int*)((char *)A + sizeof(int) * i)` ***; By doing that, the program knows both the address and the number of bytes to access for each pointer variable.***

5. For all the TODOs, I recommend you do one by one. When you complete one, compile and run to see whether you have the expected output. If the output is what is expected, move on to the next TODO.
6. Using either of the approaches for editing and develop the program, if you want to save the file for future use, you need to find a way to copy the file to a location that you can access later on.

Number Systems: Answer the following questions. Make sure to show your work. (3 points each for total 21 points).
Do NOT use a calculator of any kind, you need to know how to solve these problems by hand. If you need help, ask a TA.

1. What is the binary number 1101 in decimal?

2. What is the decimal number 234 in binary?

3. Convert the hexadecimal number A3D into binary.

4. Add 1010001 and 111111 in binary. Convert the answer to decimal. Verify your answer by first converting the binary numbers into decimal, then adding.

5. What is the largest unsigned (positive) 4-bit binary number? What is the largest unsigned N-bit binary number?

6. Convert the following decimal numbers to hexadecimal numbers.

   a. 10_{10}

   b. 14_{10}

   c. 52_{10}

   d. 845_{10}

7. How many bytes are in a KB (kilobyte)? In a MB (megabyte)? In a GB (gigabyte)?

Submission:

Please include everything in one file that include the following:

- For lab02_address_1Dstencil.c, in the submission file, please paste your completed source code and the screenshot showing its execution and output.
- For number system questions: please include in the file your answers only and number them correctly. You do not need to include the questions themselves in the answer: