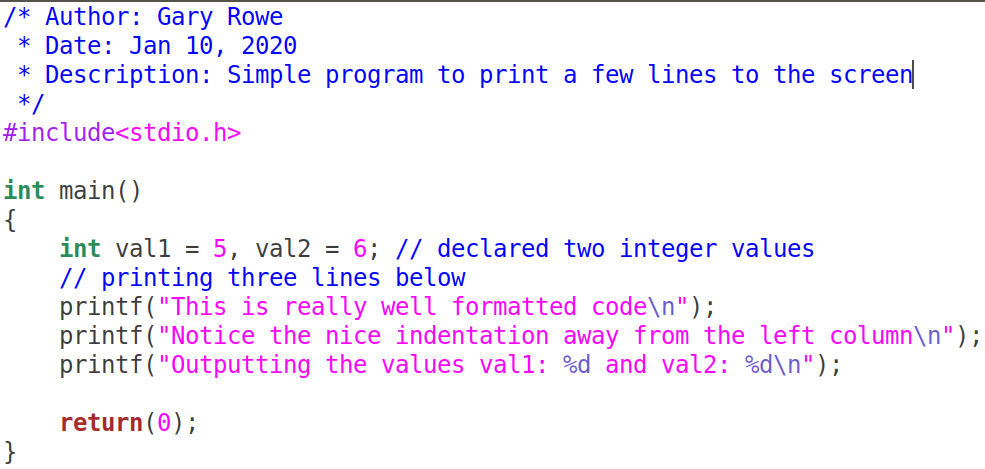
**Student Name:** **Weight: 70**

**Student ID:** **Marks: \_\_\_\_\_\_\_\_\_\_**

**CREATE AN EMPTY DOCUMENT TO SUBMIT YOUR SOLUTIONS. DO NOT USE THIS DOCUMENT TO SUBMIT YOUR ANSWERS. YOU WILL LOSE 10% FOR DOING SO!!!**

Assignment 5:

Your C files MUST be properly ***formatted*** with ***indentations*** that enhances your code ***readability***. Example of properly formatted code:

# Introduction

Up to this point we have created our own functions and seen how we could manipulate data using those functions. By creating our own functions and passing arguments when we call them, it becomes clear how flexible our programs can become.

We are now ready to do further analysis which involves understanding the structure of pre-built functions. We will get an understanding of what is expected for arguments and what the function will return assuming there is something to return.

From a rudimentary perspective we will now get some insight into the way some functions possibly operate when called. This will require a bit of imagination …

In this lab we will evaluate 6 functions provided you by the dynamic library **libc.so.6**.

Problem 1 (60):

You will use the man pages to investigate the following functions. Use the table template on page 3 to analyze each of the functions. The **example** in the table is for the function **gets**. Perform your analysis and add more rows/columns for each function.

**strcpy, strncat, strncasecmp, strstr, atoi and fgets.**

Show an example of usage of the function. If the function is also vulnerable show the unsafe usage instead. Your resources can include, the man pages, google etc.

**Please appropriately give credit to resources you use.**

Do not make your example code too long; **keep it simple**.

Keywords and datatypes to lookup before you get started:

**const, void**, **size\_t**, **FILE**

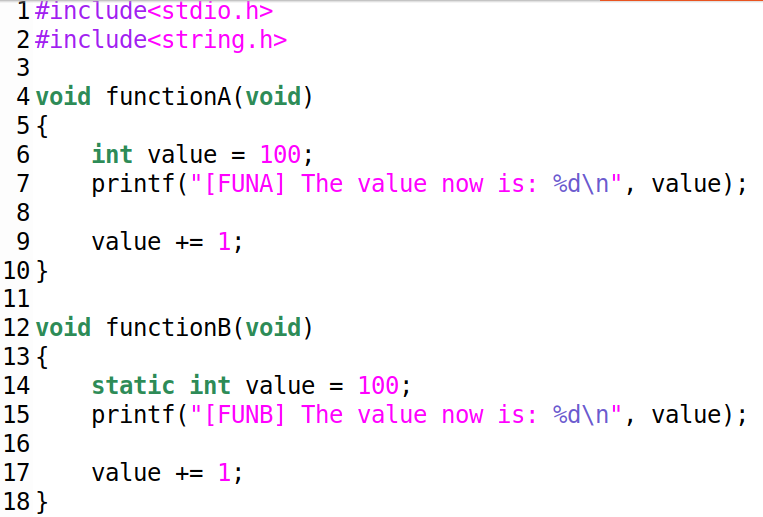
Perform your write-up in the table below. You may rework the table below or choose a different layout when you submit your answers.

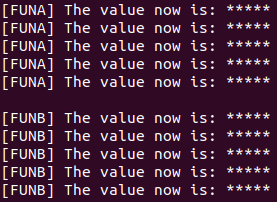
|  |  |
| --- | --- |
| Function | RESEARCH RESULTS |
| gets | **Parameters:**   * Pointer to char (holds address of array used to get input from stdin)   **Return Type:**   * Pointer to char (holds address of array used to get input)   **Example:**  char arr[10]; char \*ptr; ptr = gets(arr); **// This is a problem if the user enters more than 10 characters.** printf("%p == %p \n", arr, ptr); **// This is just show the contents of the pointer and of the &arr[0]**  **Function Bugs:**   * This function is considered to be unsafe if the user enters a string longer than 10 characters; this can cause a buffer overflow. See BUGS section of the man page.   **Location of prototype:**   * stdio.h *Prototype =>*char \*gets(char \*s); |

Problem 2 (10):

In this question you will investigate the static keyword to determine how it can be used. Below are two functions **functionA** and **functionB**. Create a main function that will call each function 5 times.

Note: Using a loop is more efficient that writing multiple lines to do the same thing.





The above images show the functions and possible output of your code.

**Note the \*\*\*\*\* are placeholder values and your program should print numbers.**

Submit your completed code and answer the following questions.

## Questions

1. Based on the behavior of the functions what can you conclude about the static keyword?
2. Will either functionA or functionB return a value? If they return a value what do you think the value will be?
3. What arguments does functionA and functionB accept? If the functions take an argument what argument will you send.
4. Modify both functions so that:
   1. They will return the content of value each time the function is called.
   2. They will save the returned value and print it in main.
5. Show the modified function in your lab report.