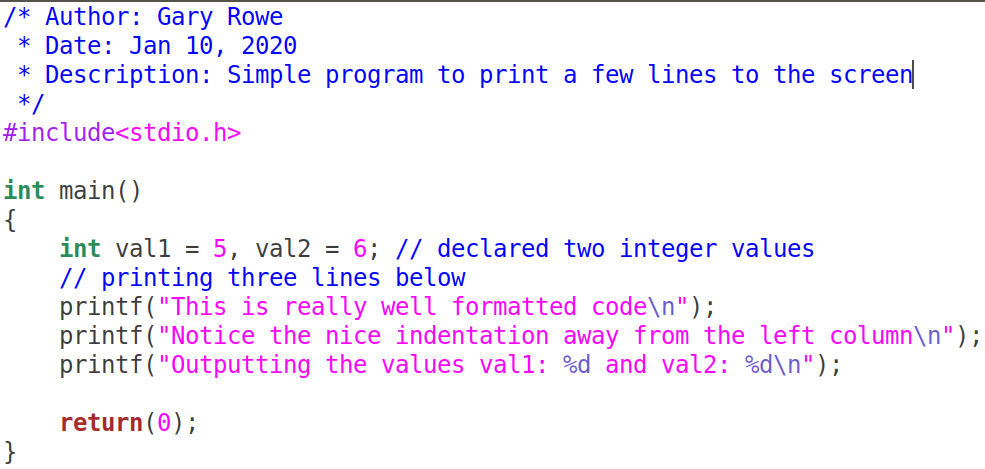
**Student Name:** **Weight:**

**Student ID:** **Marks:** \_\_\_\_/60\_\_\_

**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 4: Arrays

Your C files should be properly ***formatted*** with ***indentations*** that enhance code ***readability***. Example of properly formatted code:

**Important:**

* On your Ubuntu VM guest machine create a folder called ITSC202.
* Inside the ITSC202 folder create a subfolder called **A4**.

A4 is the folder you will use for all the C files for **Assignment 4**.

Problem 1 (20):

You have been asked by a friend to write a program for her dad that takes up to SIZE **floating** point numbers from the user. The program will then print out the sorted and unsorted values in 2 columns.

**Restrictions**: You must use 1 or 2-Dimensional arrays for this exercise. NO pointers are allowed for this exercise. **DO NOT** use global variables. **DO NOT** use library functions other than printf, scanf and fgets.

Criteria:

* On your Ubuntu VM guest machine create a folder called **a4**. This will contain all the C files for **Assignment 4**.
* Create a file called **m4p1.c**
* You will use arrays to accept SIZE numbers from user input
  + SIZE is defined as follows: **#define SIZE 12**
* You will create 3 extra functions and your main function.
  + You can place those 3 functions above main.
  + **Or**, you can place those functions below main if you use function prototypes.
* The **1st function** called **getNums** will get up to SIZE numbers from the user. It will return the number of successful entries that was made. You can test to see if the number of entries matched SIZE. Stop input when 0 is entered or SIZE numbers are typed.
* The **2nd function** called **sortNums** will sort the numbers using a sort algorithm of your choice. Since we are using a limited set of numbers, **Bubble Sort** is an ideal candidate. Do a bit of research and see how to implement it. This function will not have a return value. **Sort the numbers in ascending order.**
* The **3rd function** called **printNums** will take the sorted and unsorted numbers and print them in 2 columns, one for sorted and the other for unsorted. See the output below to get an idea.

|  |  |
| --- | --- |
| Unsorted | Sorted |
| ======= | ====== |
| 11.4 | 45.6 |
| 2.12 | 11.4 |
| 45.6 | 2.12 |
| 1.1 | 1.1 |
| : | : |

**#include<stdio.h>**

**#define SIZE 12 // Placed at the beginning of your code**

int **getNums**(/\* Your parameter list will go here \*/);

void **sortNums**(/\*Your parameter list will go here \*/);

int **printNums**(/\* Your parameter list will go here \*/);

int **main**(void)

{

// Possible variables here

// function calls here

}

float **getNums**(/\* Your parameter list will go here \*/)

{

// Contents of the function goes here

}

void **sortNums**(/\*Your parameter list will go here \*/)

{

// Contents of the function goes here

}

float **printNums**(/\* Your parameter list will go here \*/)

{

// Contents of the function goes here

}

*Above is a potential skeleton code block. Notice the prototypes are shown above main.*

Problem 2 (20):

You are working on a program that checks to determine if the user entered strings are palindromes. **What are palindromes???**

**Restrictions**: You must use 1 or 2-Dimensional arrays for this exercise. NO pointers are allowed for this exercise. **DO NOT** use global variables. **DO NOT** use library functions other than printf, scanf, fgets etc.

Criteria:

* In your folder called **A4** create a file called **m4p2.c**
* You will choose the number of strings to have the user enter.
  + You should use a **#define** constant to set the number of strings that will be entered.
* You will create 3 functions along with your main function.
  + You **WILL** place those 3 functions **after** the declaration of main.
  + You will create a header file and place all constants and prototypes in the .h file.
* The 1st function called **getStrings** will get **SIZE** words from the user, each a maximum of **20** characters wide. No return value will be made.
* The 2nd function called **isPalindrome** determines if the string entered is a palindrome. The function **isPalindrome** doesn’t have a return value.
  + **isPalindrome** will call the function **reverseStr** described below.
  + **isPalindrome** will print each and state if the word is a palindrome.
* The 3rd function called **reverseStr** will take a reference to a string of characters and reverse that word.
  + Do not use any library functions, you must build your own reverse algorithm.
  + The function **reverseStr** does not have any return value.

*Below is a potential skeleton code block. Notice there are no prototypes and the function declarations are all above main.*

***Does the location of the functions matter??? Can I place isPalindrome above reverseStr?***

**#include<stdio.h>**

**#define SIZE 12 // Placed at the beginning of your code**

void getStrings(/\* Your parameter list will go here \*/)

{

// Contents of the function goes here

}

void reverseStr(/\* Your parameter list will go here \*/)

{

// Contents of the function goes here

}

int isPalindrome(/\*Your parameter list will go here \*/)

{

// Contents of the function goes here

}

int main(void)

{

// Possible variables here

**getStrings(/\*…\*/);**

// function calls here

**success = isPalindrome(/\* … \*/);**

**// if success was x then print Palindrome successfully found x palindromes**

**// if success was 0 then print Palindrome did not find any palindrome**

}

Problem 3 (20):

Malware Analyst try to obfuscate their intentions when they write code. By doing so it makes things difficult to determine what the program’s true nature. They will often perform strange algorithms, that do not make sense.

Write a program that will read a user entered string. The string length can be maximum 30 characters long. Any printable character is allowed including the space.

Criteria:

* The program will have a function that gets the user input
* The program will have a function that takes the string and converts it to an integer number.
  + If the number is equal to 8470 the function “**challenge**” will be called. See description of challenge below.
  + The algorithm to convert the string to a number will be as follows:

Value += character \* (length of string - offset of character)

Example for word **Hello**

Value = 'H' \* (5 - 0)

Value += 'e' \* (5 - 1)

Value += 'l' \* (5 - 2) repeat for other characters.

* + If the number does not equal 8470 the function will simply return a message saying, welcome you may proceed.
    - Call **randchars** to print a number of random characters before returning to main.
  + If the function is called more than 3 times, the program will terminate without any other statement. Use exit(-1);
* The **challenge** function will randomly ask the user 1 of the available 3 riddles. The answer should be a word to keep things simple. You can make up riddles or borrow eg “Why did the chicken cross the road”.

**Question:**

1. Is this a good algorithm?
2. Explain why you think this is a good or bad algorithm.

