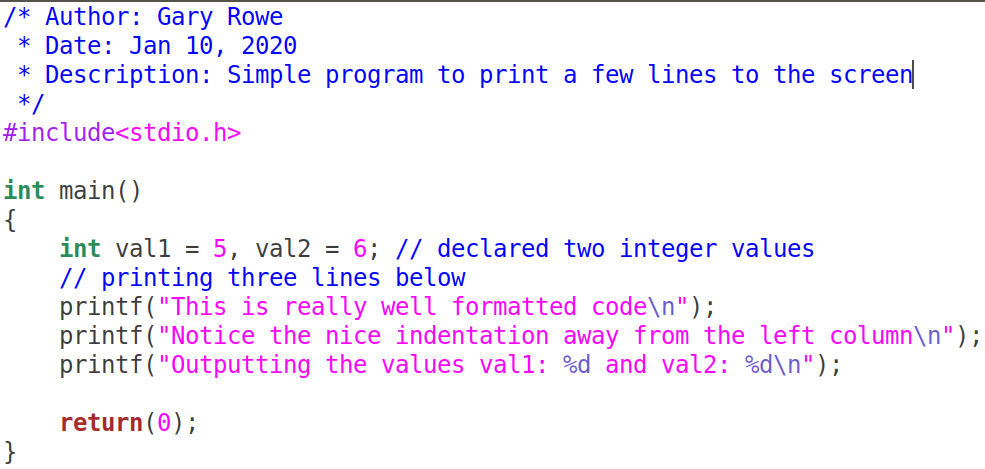
**Student Name:** **Weight: 90**

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

Assignment 7:

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

Problem 1 (25):

**Write a C program that will open the binary file numbers.txt, that contains floating point numbers, for reading. The numbers represent grades out of 100.0%.**

**It will perform the following actions:**

1. **Look for numbers below 60.00, and do the following**
   1. Open a new file called **below60.txt**, in text mode
   2. Write the position (**offset**) in the file where it was found into the new file in the following format, tab separated:

**Position: 203 Num: 23.45**

**Position: 305 Num: 34.56**

1. **Look for numbers equal to and above 60.00, and do the following:**
   1. Open a file called **above60.txt**
   2. Write the position (**offset**) in the file where it was found into the new file in the following format, tab separated:

**Position: 24 Num: 62.44**

**Position: 34 Num: 76.23**

1. **Close all files on completion and print the following information:**
   1. A count of the number of floats in the file **numbers.txt**
   2. The number of lines in the file **below60.txt**
   3. The number of lines in the file **above60.txt**

## Example output:

Numbers.txt contains 1000 numbers

Below60.txt contains 50 numbers

Above60.txt contains 950 numbers

Problem 2 (25): (Break down into manageable chunks [use functions])

**Write a C program that will open the file names.txt**

**It will perform the following actions:**

1. Ask the user to enter the first letter of a name
2. The program will open the file in read mode
3. Search through all lines in the file to find names that begin with the letter entered by the user
4. Create a filename based on the letter used to search. For example, Aname.txt would be created for names beginning with the letter A.
   1. Open the file for writing and put each name that matches the first letter criteria.
      1. If the file already exists; simply append to the file
      2. If the name already exists in the file, do not add the duplicate name.
   2. **HINT**: Investigate the function **sprintf**

Print the total number of names found in each file and only print names that are longer than 5 characters to the screen.

You can use any library functions you require. You must also modularize your code; don’t put all of your logic in main.

Problem 3 (40): (Break down into manageable chunks [use functions])

**Write a C program that will open one of the files labelled:** *filea.hex, fileb.hex, filec.hex, filed.hex, filee.hex*

**It will perform the following actions:**

1. Ask the user to enter the name of the file to open; they can choose any of the files listed above. (filea, fileb etc).

**IMPORTANT: You must ask for the filename, and use that string for opening the file.**

1. Read the contents of the file and produce the following output format

**Magic Number: 0x44424241**

**Offset Hexadecimal data format Character Format .**

00000 **41** 42 42 44 45 46 47 48 48 49 31 32 33 34 35 32 |ABBDEFGHHI123452|

00010 **4B** 42 42 44 45 46 47 48 48 49 31 32 33 34 00 22 |KBBDEFGHHI1234. . | .

* 1. **Offset** is the hexadecimal representation of the first value in each line. The value 00000 is the offset of value 41 – the letter A. The value 00010 is offset of the value 4B with which is the letter K.
  2. **Hexadecimal data format** represents the hexadecimal values of each character in the text file.
  3. **Character Format** represent the characters that were read from the file.

1. Print a ‘**.**’ when there isn’t an ASCII representation of the retrieved character from the file.
2. You should print a final summarization of the contents of the file; based on the following criteria
   1. Print the value of the first 4 bytes of the file as a hexadecimal value. **Remember that the bytes are placed in the file in little endian format.**
   2. If 75 – 100% of the bytes in the file fall in the range of printable ascii characters (0x20 – 0x7E) then conclude that this was a Text file
   3. If 25 – 74% of the bytes in the file fall in the range of printable ascii characters (0x20 – 0x7E) then conclude that this was **possibly** a Text file
   4. If less than 25% of the bytes in the file fall in the range of printable ascii characters (0x20 – 0x7E) then conclude that this is a Binary file.

## Sample output:

1:

Magic Number: 0x44424241

This is definitely a text file

2:

Magic Number: 0x44424241

This is probably a text file

3:

Magic Number: 0x44424241

This is definitely a binary file