Small Program 2

COP3223C Introduction to Programming with C — Fall 2022
Dr. Andrew Steinberg

Due Date

The assignment is due on September 19th at 11:59pm EST via Webcourses. **Do not email the professor or TAs your submissions as they will not be accepted!** This assignment is accepted late up to 24 hours with a penalty. Please see the syllabus for more information on this. Make sure to submit on time to get potential full credit. Make sure to also take into consideration the uploading time. In the past, students who are working last minute on the assignment sometimes run into uploading issues where their Internet may run slow, resulting in late submissions. The timestamp Webcourses uses for your submission will be applied and will be the final say. Please do not email the instructor or TAs saying your Internet was running slow. If the time is off by a second of the due date, then the assignment is considered late. Plan accordingly!

Important! Read Carefully! Different than Small Program 1!

This assignment contains a set of problems that are to be completed in **one C file**. You have learned about creating user-defined functions and why they are so beneficial to us programmers. For each problem in the assignment, you will create the definition of the user-defined function that is asked in the description. **If you do not create a user-defined function for each of the problems, then you will receive no credit for the problem.** Creating user-defined functions is good practice! You also must write the function prototypes! Missing function prototypes will result in points being deducted. Function prototypes are also good practice as well. The file must be named *smallprogram2_lastname_firstname.c*, where lastname and firstname is your last and first name (as registered in webcourses). For example Dr. Steinberg's file would be named *smallprogram2_Steinberg_Andrew.c*. Make sure to include the underscore character _. If your file is not named properly, points will be deducted. The script the graders use will pull your name from the file name properly. It is imperative you follow these steps so you can get your points!

Testing on Eustis

It is your responsibility to test your code on Eustis. If you submit your assignment without testing on Eustis, you risk points being deducted for things that may behave differently on your operating system. Remember, you cannot dispute grades if your code didn't work properly on

Eustis all because it worked on your machine. The Eustis environment gives the final say. Plan accordingly to test on Eustis!!

The Python Script File! Read Carefully!

A python script has been provided for you to test your code with a sample output of Dr. Steinberg's solution. This script will check to make sure your output matches exactly with Dr. Steinberg's solution file as the graders are using this to grade your assignments. The script removes leading and trailing white space, but not white space in the actual text. If there is anything off with the output (including the expected answer), the script will say your output is not correct. This includes your output producing the correct answer, however there is something off with the output display. The script does not point directly where your mistake(s) are in the code. It will only produce a success or unsuccessful output message as a whole. If you get an unsuccessful output my suggestion is to look at the sample solution text file provided to see what is different from your answer and Dr. Steinberg's when comparing. If you have extra white space or new lines or even just missing a space/new line, you will lose points that won't be changed!

Make sure you place the python script in the same directory as your C file. You can use the ls command to check to see if the following items are in the directory.

```
ls
```

You should these three files in your directory.

1. Your C File.
2. The Python Script File
3. Sample Solution Text File

If you have these three files. You are ready to run the script. Use the following command to test your code with Dr. Steinberg's provided solution sample.

```
python3 sp2test.py
```

If the script says your output is incorrect, checkout the sample text file that was generated (a new text file will be created from the script that contains YOUR output). If your numbers are off or different from Dr. Steinberg's, then that means there is something not right with code's logic and calculating the answer. However if your numbers match with Dr. Steinberg's solution, then that means there is extra/missing white space or newlines detected. Compare the text file generated by the script with solution text file line by line to find the missing/extra white space or newlines. Once you believe you found the error, rerun the script to see if the output matches.

The Rubric

Please see the assignment page for the established rubric on webcourses.

Comment Header

Make sure you place a comment header at the top of your C file. You will use single line comments to write your name, professor, course, and assignment. For example, Dr. Steinberg's header would be:

```
//Andrew Steinberg
//Dr. Steinberg
//COP3223C Section 1
//Small Program 2
```

Missing a comment header will result in point deductions!

The Solution Text File

You are provided a solution file that was created by Dr. Steinberg's Python Script. Now you may notice some strange things about the file. In this assignment, you are going to write statements that involve interacting with the user. Now you are probably wondering from the solution text file where the interaction is happening? The fact is that the Python script handles the interaction. The script creates an input stream that feeds it input. That is why you don't see the input directly in the text file. For example, the text file on the third line says `How many hours will you keep your car parked here> Car will be parked for 9 hours...` Here this looks like we should be typing input, however the python script has already fed it input. That is why you don't see the values except for the results. In each problem, a screenshot of the C file being executed manually without the Python script shows how it looks on a normal run. Carefully look at the output in the pictures provided. Note: The arrow symbolizes that the text wrapped onto the next line of the pdf file. In the text file itself it is actually one whole line.

`samplesolutionsp2.txt`

```
Enter the radius: Enter the height: The total surface area of the cone is 4432.16
Welcome to the Parking Garage!
How many hours will you keep your car parked here> Car will be parked for 5 hours
↪  and will be charged $21.05.
Enter a year after 2016: Predicted Wakanda's population for 2022 in thousands:
↪  76.885
Enter the value for n: 12! is approximately 478858054.1927
```

Problem 1

Write a user defined function definition called `coneSurfaceArea`. This function calculates the surface area of a cone. The formula for calculating the surface area is described below:

$$SA = \pi \times r \times (r + \sqrt{h^2 + r^2})$$

The function has three parameters. The first parameter is the radius of the base of the cone which can be represented as a double type value. The second parameter is the height of the cone which is also double type. The last parameter is the value of pi (defined as $\pi = 3.14$) which is also double type. For this problem you will need to collect input from the user inside the main function of your program. After collecting the input, you will then call the user defined function to have it perform the calculation and display the result up to two decimal places. The function does return the result and displays it in the main function. Make sure your output matches as expected from the following output sample in the figure. Do not worry about invalid input. We have not covered conditions yet.

```
Enter the radius: 17.4
Enter the height: 61.3
The total surface area of the cone is 4432.16
```

Figure 1: Sample output for question 1. Make sure your output matches exactly for the test script.

Problem 2

A parking garage charges customers to park their car at a rate $4.21 per hour. Write code that welcomes the users and prompts them to enter the amount hours they plan to leave their cars parked. The program will display the number of hours and the amount charged to the user. The system deals with whole number hours only. The monetary value displayed has to include $ and two decimal places. Check out the following figure for sample output that the script expects. All of this is to be completed in a user defined function called `parkingCharge`. The function will not return anything. Do not worry about invalid input. We have not covered conditions yet.

```
Welcome to the Parking Garage!
How many hours will you keep your car parked here> 5
Car will be parked for 5 hours and will be charged $21.05.
```

Figure 2: Sample output for question 2. Make sure your output matches exactly for the test script.

Problem 3

After studying the population growth of the area Wakanda in the last decade of the 20th century, we have modeled Wakanda's population function as

$$P(t) = 51.451 + 4.239t$$

Where t is years after 2016, and P is population in thousands. Write a user defined function called `wakandaPopulation` that predicts Wakanda's population in the year after 2016 and displays it to the user. The user defined function does not return a value. The user defined function also takes one parameter argument of type int called year, which is the year after 2016. Inside the main function, you will ask the user for the year after 2016. Do not worry about invalid input. We have not learned conditions yet. Check out the following figure for the sample output that the script expects.

```
Enter a year after 2016: 2022
Predicted Wakanda's population for 2022 (in thousands): 76.885
```

Figure 3: Sample output for question 3. Make sure your output matches exactly for the test script.

Problem 4

For any integer $n > 0$, $n!$ is defined as the product $n * n - 1 * n - 2 \ldots * 2$. $1!$ is defined to be 1. It is sometimes useful to have a closed-form definition instead; for this purpose, an approximation can be used. R.W. Gosper proposed the following such approximation formula:

$$n! \approx n^n e^{-n} \sqrt{(2n + \frac{1}{3})\pi}$$

For this problem, define a user defined function that performs the calculation. The user defined function takes one argument of type int and will return the result of the computation of type double to display in the main function. The result is displayed up to four decimal places. Name the user defined function `factorialApprox`. Use the same value of π from problem 1. Do not worry about invalid input. Check out the following figure for the sample output that the test script expects. **Note: You don't need to define Euler's Number. Use the math library's natural exponent function.**

```
Enter the value for n: 12
12! is approximately 478858054.1927
```

Figure 4: Sample output for question 4. Make sure your output matches exactly for the test script.