Thicccque
November 5, 2018
Dr. Parham-Mocello

Assignment 4 Part 1

Orientation:

1. Two-Year Plan (CS Major, Computer Systems Option):

First Year (2018-19)					
Fall		Winter		Spring	
CS 160	3	BI 212H	4	MUS 108	3
MTH 231	4	COMM 114	3	CS 261^	4
ALS 199H	2	CS 162^	4	MTH 306	4
HC 407	2	MTH 254^	4	PH 211+221*	5
HHS 231	2	HC 407	1	HC 407	1
MUS 347	2	MUS 347	1		
PAC 314	1				
Total:	16	Total:	17	Total:	17

Second Year 2019-20					
Fall		Winter		Spring	
ENG 202+215*	4	ED 216	3	HC 407	1
ST 314	3	WR 214	3	Pro School?	3
ECE 271+272*	4	CS 290^	4	Pro School?	4
PH 212+222*^	5	ST 314	3	Pro School?	5
HC 407	1	HC 407	1		
Total:	17	Total:	14	Total:	13

* - Has a corequisite (listed in same cell of table).
^ - Has a prerequisite course (all courses are listed in proper order).

This plan is not fully complete as of now (as clear by the one certain credit in the Spring term of 2019-2020), but that is because I need to meet with a College of Science advisor to discuss the possibility of a Computer Science and Mathematics double major with them, and I also need to understand how Pro School works or will work when I get there; before I can fill in that term in good conscience. I chose this path because it fulfills all of my Bacc Core requirements (besides the synthesis requirements) and all of my pre-CS requirements as well. The two following years are completely open for major requirements or a second major / minor. That is why I chose this path for these first two years.

Computation:

1. Understanding the Problem / Problem Analysis:
    - Do you understand everything in the problem? List anything you do not fully understand, and make sure to ask a TA or instructor about anything you do not understand.

        Yes, I think I do understand everything in the problem. I had been confused whether or not the program needs to handle any user-inputted function, which would very much complicate the problem and the complexity of the required solution, but Dr. Mo has since clarified that the assignment only needs to handle f(x) = x^2, although allowing for an easy modification of this function would be preferable. The other question is whether the rectangular riemann sum we are performing should be left-handed, mid-point, or right-handed, so we'll just need to assume one of them.

    - What are the user inputs/requirements, program outputs, assumptions, etc.?

        User Inputs / Requirements:
        - Ability and willingness to input strings / integers to signify their preferences at different decision / information gathering moments.
        - User's desired calculations (integral or summation approximation) or if they would prefer to exit the program.
        - User's desired bounds on which to integrate / summate.
        - User's desired number of rectangles / trapezoids for integration (accuracy).

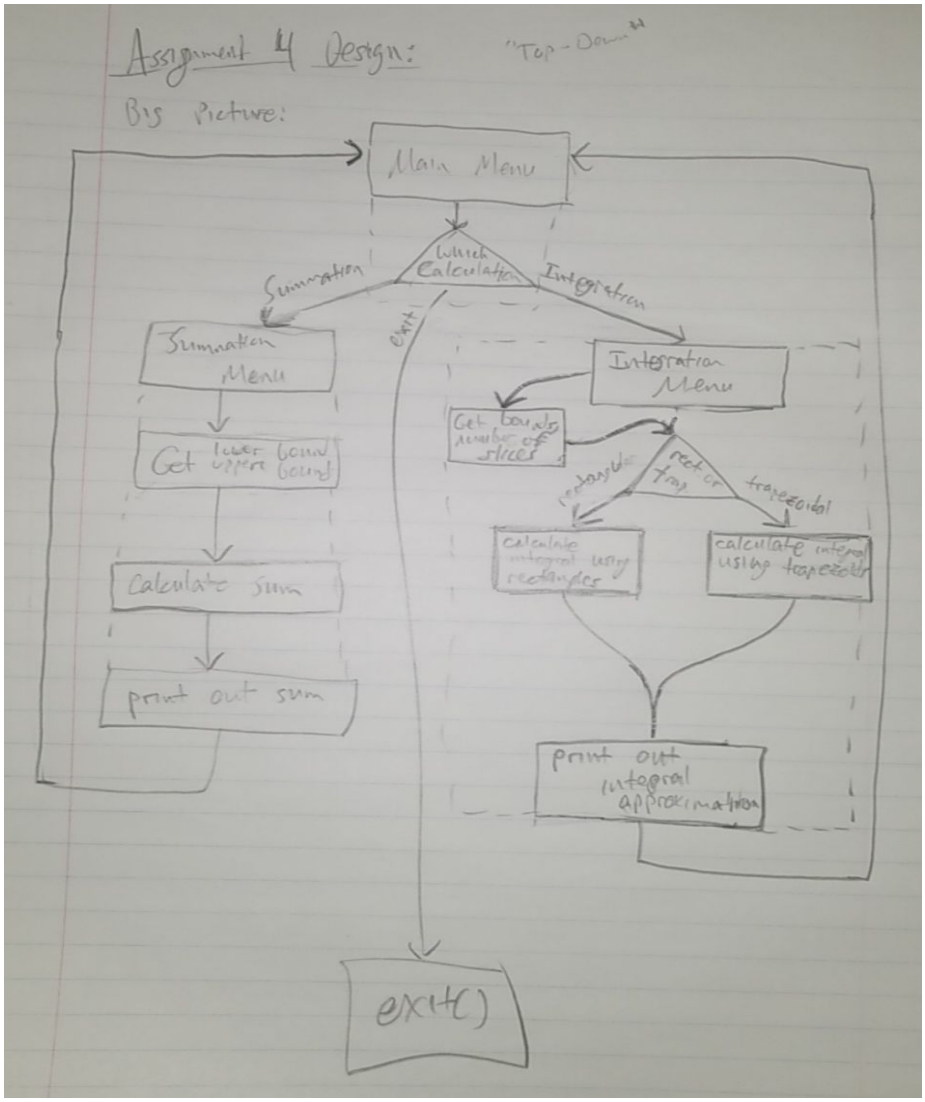
        Program Outputs:
        - The results of their calculations, formatted to apply to either summation, trapezoid integration, or rectangle integration approximation, depending on user choice. Printed to console.

        Assumptions:
        - Summation will be approximated by summing the values of f(x) for every integer between the given bounds, not any more accurate.
        - The bounds used for either calculation should be integers, as it is not clear in the instructions if the calculations should be able to be floating point numbers. They can also be either positive or negative without error.
        - Integral approximation using the rectangle summation will be left-handed, as it is not clear in the instructions if the program should be able to handle any others.

        Functional Requirements:
        - Check whether a given string is an integer or not without throwing exceptions.
        - Get input from the user, only allowing execution to continue if the input is valid (usually must be an integer).
        - Evaluate the function that will be integrated or summated (coded as f(x) = x^2 to begin with) for any numerical value of x.
        - Get user input to set the bounds of integration / summation and/or the number of rectangles / trapezoids for the calculation of the approximation.

- Calculate rectangular approximation of f(x) on any interval with any number of rectangles, without throwing any exceptions.
- Calculate trapezoidal approximation of f(x) on any interval with any number of trapezoids, without throwing any exceptions.
- Calculate summation approximation of f(x) on any interval provided, without throwing any exceptions.
- Print result of the user's desired calculations, appropriately formatted, to the console.
- Return to the main menu after completing a calculation.

2. Program Design:
   - What does the overall big picture of this program look like?



   - What are the decisions that need to be made in this program?
     - Which mode to send the user to (summation or integration approximation).
     - Which method of integration approximation (rectangle or trapezoid approximation).
     - Whether the user exits the program after making a calculation (based on user input).

- What are all the tasks the program must complete to succeed?
  - A function that returns f(x) when passed an x.
  - A function that checks whether a passed value is an integer.
  - A main menu function to send the user to their chosen calculation or exit.
  - An integration menu function to get the bounds and desired method of calculation.
  - A summation menu function to get the user's input for the bounds of the calculation.
  - A function to calculate the summation approximation.
  - A function to calculate the integration approximation using rectangle method.
  - A function to calculate the integration approximation using trapezoid method.

- Based on these tasks, what do the algorithm details in each task look like? What are the decisions you'll make within each task? How and where are you going to handle bad input? (Pseudocode)
  - f(x): (passed number 'x')
    Return x to the power of two (x**2)

  - Check if Integer: (passed a string 'input')
    For each character in the string:
      If not in the ASCII range of digits or starting with character other than negation):
        Print that the input is not an integer
        Return false (that it is not an integer)
    Return True (none of the characters were invalid)

  - Main Menu:
    In a loop until inputs are all valid:
      Prompt user for desired mode (integration, summation, or if they want to exit)
      User_input = this answer
      If user input was a valid integer (using prior function):
        Break loop, continue execution
    If User_input is integration:
      Send user to Integration Menu
    Else if User_input is summation:
      Send user to Summation Menu
    Else if User_input is to exit:
      Exit program

  - Summation Menu:
    In a loop until inputs are all valid:
      Prompt user for lower and upper bounds of summation
      Lower_bound = the user input for this
      Upper_bound = the user input for this
      If user inputs were all valid integers (using prior function):
        Break loop, continue execution
    Send user to summation calculation function (following)

- Integration Menu:
  In a loop until inputs are all valid:
    Prompt user for bounds and number of "slices" (accuracy) of the integration
    Store inputted values as variables
    Prompt user for desired method of integration (rectangle or trapezoid)
    If all user inputs were valid integers (using prior methods):
      Break loop, continue execution
  If desired method is rectangle approximation:
    Send user to rectangle calculation, passing user inputs
  Else if desired method is trapezoid approximation:
    Send user to trapezoid calculation, passing user inputs

- Summation Calculation: (passed lower and upper bounds)
  For each integer between lower and upper bounds (inclusive):
    Result += f(x) of current integer (by calling the f(x) function)
  Print out Result

- Rectangle Integration Calculation: (passed bounds and number of rectangles ["slices"])
  Initialize a variable Result at 0 to later hold the result of this calculation.
  Width_of_rectangle = (upper bound - lower bound) / number of rectangles
  For each rectangle of this width in the interval:
    Height_of_rectangle = f(x) of the lower bound + the prior rectangles widths
    Result += Width_of_rectangle * Height_of_rectangle
  Print out Result

- Trapezoid Integration Calculation:
  Initialize a variable Result at 0 to later hold the result of this calculation.
  Width_of_trapezoid = (upper bound - lower bound) / number of trapezoids
  For each trapezoid of this width in the interval:
    Height_1 = f(x) of the lower bound + the prior trapezoid widths
    Height_2 = f(x) of the lower bound + the prior and current trapezoid widths
    Average_height = (Height_1 + Height_2) / 2
    Result += Average_height * Width_of_trapezoid
  Print out Result

- The flowchart for the overall flow and the pseudo code for each of the individual operations referenced in that overall flowchart are listed prior. Together, these constitute a full plan.

3. Create a test plan with the test cases. What do you hope to be the expected results?

Test Plan			
Variable	Case	Input	Expected Result
Desired calculation mode	Good	"S"	Send user to the summation menu
		"I"	Send user to the integration menu
		"EXIT"	Exit the program
		"s"	Send user to the summation menu
		"i"	Send user to the integration menu
		"ExIt"	Exit the program
	Bad	"1"	Get re-input
		"gashnisdb"	Get re-input
Lower / Upper Bounds	Good	-5	Use the value for calculations
		0	Use the value for calculations
		5	Use the value for calculations
	Bad	0.5	Get re-input
		"asnas"	Get re-input
	Edge	<empty field>	Get re-input
		100000000	Use the value for calculations
Number of "slices" (number of rectangles / trapezoids)	Good	-5	Use the value for calculations
		0	Use the value for calculations
		5	Use the value for calculations
	Bad	0.5	Get re-input
		"asnas"	Get re-input
	Edge	<empty field>	Get re-input
		100000000	Use the value for calculations

4. Extra Credit: List courses by the term (Fall, Winter, Spring) in your 2-year plan. This requires looking at the schedule of classes to know when courses are being taught.

5. Extra Credit: Implement the two error functions to determine if a value is a good integer or good floating point number and the summation function.

- Check if input is a good integer:

```
def check_if_integer(string_input):
    for index in range(len(string_input)):
        character = string_input[index]

        if(
            not((character >= '0' and character <= '9') or
                (len(string_input) > 1 and index == 0 and character == '-'))
        ):
            print('Error! Input is not an integer!')
            return False

    return True
```

- Check if input is a good float:

```
def check_if_float(string_input):
    count_of_points = 0
    for index in range(len(string_input)):
        character = string_input[index]

        if(character == '.'):
            count_of_points += 1

        if(
            count_of_points > 1 or
            not((character >= '0' and character <= '9') or
                (len(string_input) > 1 and index == 0 and character == '-'))
        ):
            print('Error! Input is not an float!')
            return False

    return True
```

- Calculate summation:

```
def summation_calculation(lower_bound, upper_bound):
    result_of_approximation = 0

    for x in range(upper_bound - lower_bound + 1):
        result_of_approximation += f(lower_bound + x)

    return result_of_approximation
```