CSC190: Computer Algorithms and Data Structures
Assignment 1
Assigned: Jan 15, 2017; Due: Feb 3, 2017 @ 10:00 a.m.
To be Completed **Individually**.

1 Objectives

In this assignment, you will implement the Simpson's method and the adaptive Simpson's method to compute the definite integral of a function (i.e. integration over finite real interval). When directly computing the integral is difficult, numerical integration methods are useful alternatives as these allow you to approximate the result with certain degree of precision. There are three parts in this assignment and each part builds upon the previous parts as follows.

1. You will write a simple function which will be used for evaluating a continuous mathematical relation using existing built-in functions in the `math.h` library.
2. You will implement a function for performing numerical integration of the relation constructed in the previous part using the Simpson's method.
3. You will construct another function that performs integration in an adaptive manner in order to reduce the magnitude of error introduced by the general Simpson's method.

This assignment tests your understanding of basic C syntax (i.e. arrays, loops, selection statements, etc.). As this is your first assignment, we have provided files that you must use as a starting point to complete this assignment. You should download the content of the folder `Assignment1` which contains two subfolders (`code` and `expOutput`) into your ECF workspace. Folder `code` contains a skeleton of function implementations and declarations. Your task is to expand these functions appropriately. `main.c` evokes all the functions you will have implemented and is similar to the file that will be used to test your implementations. Use `main.c` file to test all your implementations. Folder `expOutput` contains outputs expected for the supplied `main.c` file. Note that we will **NOT** use the same sample files for grading your assignment. Do **NOT** change the name of these files or functions.

2 Grading

It is **IMPORTANT** that you follow all instructions provided in this assignment very closely. Otherwise, you will lose a *significant* amount of marks as this assignment is mostly auto-marked and relies heavily on you accurately following the provided instructions. Following is the mark composition for this assignment (total of 20 points):

- Successful compilation of all program files i.e. the following command results in no errors (2 points): `gcc a1Part1.c a1Part2.c a1Part3.c main.c -lm -o run`
- Output from Part 1 and 2 exactly matches expected output (2 point each for a total of 4 points)
- Output from Part 3 exactly matches the expected output (4 points)
- Code content (10 points)

Sample expected outputs are provided in folder `expOutput`. We will test your program with a set of completely different data files. No late submissions are accepted. All submissions after the due date will be assigned a grade of 0/20.

Part 1: Evaluating a continuous function

This part requires you to expand the function `f` located in the `a1Part1.c` file.

- One argument `x` of type `float` is passed to this function.

- This function will return a value of type `float`.
- The polynomial you will implement is $f(x) = sin(x) + x^{10} + (x-1)^2$
- Please refer to the file `Part1.txt` for a sample result from this function.
- You can test your implementation of this function using the provided `main.c` file. As you are aware from going through the preparation slides, in order to execute this program, all associated functions must be compilable. Hence, ensure that before you test this function that other skeleton functions are at bare minimum compilable. Then, compile all files using the command supplied in Sec 2. When you examine the `main.c` file, you can observe that you must supply to the executable file the argument '1' to invoke the `f` function (i.e. execute the program with `./run 1` after compilation).

Part 2: Implementing Numerical Integration using Simpson's Method

This part is a continuation of Part 1. Here you will implement the function `S` located in `a1Part2.c` which numerically integrates the function `f` in the interval $[a, b]$ (i.e. $\int_a^b f(x)$). Numerical integration is used in solvers such as MATLAB in order to evaluate the integral of a function within a specified interval. The Simpson's method approximates the integral of a function by evaluating the following expression:

$$\int_a^b f(x) \approx \frac{b-a}{6}(f(a) + 4f(m) + f(b)) \tag{1}$$

where m is the mid-point of the interval $[a, b]$. The Simpson's method approximates the function f using a third order polynomial. The error in this approximation is proportional to the fourth derivative evaluated at $\varepsilon \in [a, b]$ (i.e. $f^4(\varepsilon)$). Hence if f is a third order polynomial, there will be no error introduced by this numerical integration.

- Two arguments are passed to the function and these are the bounds of integration (i.e. [a,b]).
- The function will evaluate and return the approximate value of $\int_a^b f(x)$ in accordance to Eq. 1.
- `main.c` file can be used to test your implementation via the command `./run 2`. Ensure that your output matches the sample output provided in `Part2.txt`. You can integrate the expression f by hand. Compute the correct value and check how different the result from this function is from the actual value.

Part 3: Adaptive Numerical Integration

As you may have noticed in the previous section, the error in applying the Simpson's method over a large interval is significant. Here you will implement an adaptive version of the Simpson's method which splits the original interval into smaller halves until the error within each divided interval is lesser than the threshold ϵ. Then, the Simpson's method is applied to each one of these intervals and the results are summed. You will adaptively implement this as functions may have high variations in one area and not vary so much in other regions. It is necessary to divide the intervals extensively in the highly varying regions and not so extensively in slowly varying regions. The original interval will be continually divided into half until the current interval satisfies:

$$|S(a, m) + S(m, b) - S(a, b)|/15 < \epsilon$$

where $S(a, b)$ is the application of the Simpson's method to the interval $[a, b]$ and m is the midpoint of the interval $[a, b]$. At this point, $S(a, b)$ estimates the integral over the sub-interval $[a, b]$ with sufficient accuracy and there is no need to further divide $[a, b]$. Hence, once the error over $[a, b]$ satisfies the above threshold, $S(a, b)$ can be applied to obtain the estimate of the integral over $[a, b]$ (i.e. **no** need to use $S(a, m)$ and $S(m, b)$). This is repeated throughout the entire original interval. You will implement the function `simpsonsAdaptiveIntegration` located in `a1Part3.c` as follows:

- Four arguments are supplied to this function and these are the interval of integration $[a, b]$, ϵ (for the stopping criteria) and the length of the smallest interval allowed (in order to prevent too many iterations).
- This function will be implemented **iteratively** (loops NOT recursion). In the iterative implementation, you will utilize a loop and an array to aid with the iterative division of intervals and evaluation of the integral over these sub-intervals. Note: we will check whether you have used recursion (you will lose all points for this part and code content if this is the case).
- Define a macro called `ARRAYSIZE` in the `a1.h` file which is the size of the array that you will declare in the function `simpsonsAdaptiveIntegration` that will hold information about the intervals that are yet to be evaluated.
- This function will not return any values. It will print the result of applying the adaptive Simpson's method. Please refer to the file `Part3.txt` for the expected formatting of your output.
- `main.c` file can be used to test your implementation via the command `./run 3`. Ensure that your output matches expected output in `Part3.txt`. You can integrate the expression f by hand. Compute the correct value and check how different this result is from the output of the adaptive method.

3 Code Submission

3.1 Checklist

ENSURE that your work satisfies the following checklist:

- You submit before the deadline.
- All files and functions retain the same original names.
- Your code compiles without error in the ECF environment (if it does not compile then your maximum grade will be 1/20).
- Do not resubmit any files in Assignment 1 after the deadline (otherwise we will consider your work to be a late submission).

3.2 Submission through `submitcsc190s`

- Log onto your ECF account.
- Ensure that your completed code compiles.
- Browse into the directory that contains your completed code (`a1.h`, `a1Part1.c`, `a1Part2.c`, `a1Part3.c`).
- Submit by issuing the command:
  `submitcsc190s 1 a1.h a1Part1.c a1Part2.c a1Part3.c`