GNG1106 Fall 2016 – Project Deliverable 2

Available: Nov 20, 2016
Due: Nov 27, 2016

Instructions

This project deliverable is to be done in teams of 2. Follow the instructions in the GNGBlackboard document that describes how to submit assignments through the Blackboard. The following are specific instructions for these deliverables:

- You will need to submit your deliverable electronically to Blackboard.
- The project document is provided in both PDF and Word file (Word, Rich Text Format). For deliverable 2, complete Section 3 (added to your sections from the first deliverable). Submit the complete document that is the title page and sections 1 to 3.
- You must submit a type written document. Hand written documents are not allowed.
- Submit the PDF document (other versions, including Word files shall not be accepted).

- Do start the deliverable soon and do **not** wait until the last minute. You will be more efficient with a number of smaller efforts over a few weeks before the deadline than one large effort just before the deadline.
- Coordinate and share the work with your partner. Communicate regularly. Meet to discuss the work. Review your partner's work. It is important that you know the complete design, not only the part you completed.

Marking Scheme

- See the Rubrics marking scheme provided separately.
- The marking of each deliverable is based on 30 total marks.
- Each of the two deliverables count for 5 marks of the total project mark (i.e. 5/25). Thus both deliverables which are used to develop the software design count for 10 marks of the total project mark (i.e. 10/25).

The following shows how each deliverable contributes to the final project mark.

Deliverable	Problem Methodology Step	Project Document Sections	Mark
Deliverable 1	Step 1 (Problem Identification and Statement)/Step 2 (Gathering of Information and Input/Output Description)	Sections 1/Sections 2	5
Deliverable 2	Step 3 (Test Cases and Software Design)	Section 3, Revision of Sections 1 to 3	10
Deliverable 3	Step 4 (Implementation)/Step 5 (Software Testing and Verification)	C Source Code, Sections 4/5, Revision of Sections 1 to 3	10
	Total Mark		25

Guidelines

The following Guidelines will help you complete this deliverable.

- The objective of this deliverable is to provide the design of the software. You shall also revise Sections 1 and 2 of the project document according to feedback provided. Do discuss with your partner and contact the TA or professor to answer any questions.
- Use the project document provided as the template for completing this deliverable (do not include sections 4 and 5 in this deliverable).
- Consider test cases that vary the range of the input values. Present a minimum of five test cases. Test cases that use similar input values are not useful.
- Note that the Design section consists of a number of subsections that divides the presentation of your functions according to their role in your software. For each of these subsections:
    - Provide an introduction – text that describes the overall role of the function presented.
    - For each function, present a design using the format presented on the next page. Ensure that your presentation is consistent.
- All projects require that up to 5 sets of input values can be stored in a file and reselected by the user when starting the program again. Here are a few hints:
    - Ensure that your software is designed to check the user input values (Section 2 of the project should provide information on valid values for user input).
    - Use a structure whose members contains all user input values.
    - Define an array of 5 structure values.
    - The array can then be written with a single `fwrite` function call to a binary file and read from the binary file using a single `fread` function call.
    - Add a member to the structure to indicate if the structure element in the array has been filled (i.e. is TRUE when filled, and FALSE when free to accept values).
    - Define a file name (e.g. file.bin, do choose a more significant name than file) as a symbolic constant: `#define BINFILE "file.bin"`. You can then use `BINFILE` with `fopen` as in: `fp = fopen(BINFILE, "wb");`
    - When starting the program check if the file exists (try to open the file for reading).
        - If it does not exist, then create a new file and save the structure array (with all elements in the array initialized to available) into the file. Then ask the user for new input values.
        - If the file exists, list the contents of each element in the array (with some message like: "Available" for elements that do not yet have values). Prompt the user to see if one of the existing values is to be used or to select new values
    - When a new set of values is given by the user, prompt the user to save the values. If an element in the array is available, update it with the new values; otherwise prompt the user to replace an existing set of values.
    - The above logic is a suggestion to help you get started. Do consider alternatives.

Function Design

The format for presenting the design of a function (an example is shows below) starts with the function header that lists all parameters, followed by a description of each parameter, a description of the return value (if one is returned), and then a description of the logic/algorithm of the function. The logic description should provide enough detail so that the coding is straightforward (i.e. does not require much thinking on the logic of the function).

Example of Function Design

findAllRoots(double start, double end, int n, double coefficients[], double roots[])

Parameters:

- start: gives the start of the interval for searching for roots.
- end: gives the end of the interval for searching for roots.
- n: gives the number of polynomial coefficients in the array referenced by "coefficients".
- coefficients: references an array containing the polynomial coefficients. The parameters n and coefficients parameters defines the polynomial.
- roots: references an array for storing the values of the roots found for the polynomial.

Return value

The number of roots found is returned. This value also gives the number of values that are stored in the array referenced by roots.

Logic/Algorithm

This function finds all the roots on the interval defined between start and end. The function first defines a subinterval size as `subinter = (end-start)/SI_RESOLUTION`. A determinant loop, using `cntr` (`cntr` is varied from 0 to `SI_RESOLUTION`-1), is then used to evaluate each subinterval as follows:

- The limits of the subinterval is defined with
  `ak = start + cntr*subinter)` and
  `bk = start + (cntr+1)*subinter` (if `bk` is greater than end, it is set to `end`)
- The function `findRoot` is called to see if the subinterval contains and if it does,
  - the root is saved in the `roots` array at index `rootIx` and
  - `rootIx` is incremented.

The index (which is incremented after saving the root in the roots array) used to save the root value in the array after the loop is finished contains the number of roots saved in the array. Its value can be returned by the function to give the number of roots found and saved.