Assignment Unit 3 User-Defined Data Types

1 Overview

The task is to implement a user-defined data type plus related functions.

This programming project is to be undertaken in groups of three to five students - please add your groups to the "Wiki" section of the module's SULIS page.

Make sure to distribute your code over suitable files. Source code must be properly formatted and commented. Projects must compile (using gcc) and run on Ubunut Linux. Solutions are to be submitted via SULIS as a single tar archive (please make sure before submission that all required files are inside the tar archive - if you archive is incomplete, you will loose significant marks).

Deadline for submission of your project solution is Friday of week 12 (28.11.2018).

Provide an data type for a polynomial of arbitrary order, i.e. your ADT should be able to store the coefficients of a polynomial of the form $a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0$, where the order/degree n of the polynomial is determined at run-time.

To facilitate arbitrary order of the polynomial, **store all the coefficients** of your polynomial in a variable length array (VLA - cf. slide 108 of Unit 1 Intro to C) or a dynamic data structure such as a dynamic array or a linked list. Do **not use** "normal" static arrays - their size needs to be defined at compilation time and therefore they are not suitable.

Hint: In order to simplify the operations on polynomials, you should store all coefficients from a_0 to a_n, i.e. the stored coefficients of the polynomial $x^4 + 3.1x^2$ are $a_4 = 1,\ a_3 = 0,\ a_2 = 3.1,\ a_1 = 0,\ a_0 = 0$.

Provide at least the following operations:

- create/delete a polynomial
- add/subtract two polynomials
- multiplication/division by a double (mulitply/divide each coefficient with/by the double)
- normalise polynomial (adjust all coefficient so that the coefficient of highest order term = 1)
- return order of polynomial (the highest power with coefficient $\neq 0$)
- print polynomial to stdout

Please follow the instructions in the Design section closely.

2 Design

- Design a suitable `struct` data type for your polynomial - i.e. decide on how you plan to store polynomials.
- Feel free (not required) to `typedef` you struct to a more convenient name.
- For each operations, write down a specifications as discussed in the Seven Steps of Systematic Program Construction.
- Then decide how the user will see your operations and write down function prototypes (function declarations) for all operations.
- For each operation, develop an algorithm in pseudo-code, test it (desktop testing) and repeat this until you are happy that your algorithm is working correctly.
- Implement each operation in the C programming language.
- Once your data type and all functions are implemented, design a test main-function (in a separate file) to verify that your implementation complies with your specification. Keep in mind, that testing is a destructive process, i.e. the tester does his best to break the program. If the tester cannot cause your program to crash on purpose, you can be confident, that a user will not crash your program by accident. Pay particular attention to border cases and illegal values for parameter.
- You must provide your own Makefile (this must be written by yourself - no IDE generated Makefile will be accepted). Your Makefile must have the following features:
  - Use separate targets for compilation and linking.
  - Use at least one make file variable.
  - Provide a "clean" target that removes temporary files, object files and executables created by this Makefile.
  - Does **not use any implicit rules**.
  - Running make without a target should compile & link your application and also automatically execute your application.

3 Deadline & Marking

This project is worth 20% of the module.The **Deadline** for the project is Friday of week 12 (30.11.2018). In general, all group members will receive same marks - if any member is not contributing, please contact me asap (marks may be reduced).

Please submit your solution as a single tar archive via the SULIS page (only one submission per group is needed). To create a tar archive, use the command `tar cvf projectArchive.tar projectFolder`, where `projectArchive` is the name you want to give to your archive and `projectFolder` is the directory that contains **all** your deliverables (you can enter "`man tar`" to read information about tar - press 'q' to quit). Please make sure:

- **not** to compress you archive, as this sometimes causes problem when extracting (do not use the 'z' option or similar).
- you are using tar and no other archieve tool.
- you are using a ".tar" extension.
- your archieve contains **all** deliverables - I cannot accept any modifications to your submission after the deadline.

A complete solution includes the following:

1. A text document containing an explanation of your designed data type, your specification and your pseudo-code for all operations.
2. Well formatted and documented source code - distributed over suitable files - that implements the project.
3. A test main function that tests all the operations on your data type.
4. A Makefile used to compile the application (as outlined in the design section)

The marking scheme for this project is as follows:

Data Type explanation:	3
Specification for all operations:	3
Pseudo-Code for all operations.	4
Well formatted source code for your data type and all operations (distributed over suitable files)	7
Test Main Function	4
Makefile	2
Penalties:	
Insufficient comments:	Up to -30%
Poor code format:	Up to -30%
Poor file structure	Up to -15%
Bad coding style (e.g. using `goto` or global variables)	Up to -50%
Total	**20**