Lab session (23rd Feb)

1. Easy

1.1. [string] Implement your own string type using struct. Support functions on strings such as string length, substring, concatenation, and trim. In the struct that you use, store and keep track of string length under all operations. (Trim(str, i) should remove the last i characters of the string).

1.2. [enum names using array of strings] Create an enum. Also create an array of char *'s where the "names" of the enums are stored. Create supporting routines that take the enum value and print the enum name. Learn, and use *switch-case* for the latter.

1.3. [Changing default values in enum] Change the enum in 1.2 by assigning integer values, in non-increasing order, to each entry in the enum. Print the enum names.

1.4. [typedef] Use typedef to create a shorthand name for your created enums and structs.

2. Normal

2.1. [Complex number multiplication] Take two complex numbers as input, and output the product of them. All complex numbers should be stored in a struct.

2.2. [Use of malloc and -> operator on structs] Learn the syntax to allocate structs using malloc. Also, learn to use the -> operator of C, by which one can access a field of a pointer to a struct.

- Use the complex number examples that we studied in the class.

2.3. [Stack] Implement a stack using struct and functions push and pop. The max stack size can be specified during stack creation. Keep track of the top of the stack and size of the stack as variables inside your struct. Fix conventions to indicate pop from empty stack and push to full stack. Alternately, use realloc to allow for expansion of the stack limit. Use the *string* that you designed in 1.1.

3. Learn by Experiments

3.1. [AoS and SoA] Define an array of structs. Also define a struct of arrays. Find out the advantages of both.

3.2. [Academic student records as structs] Design structs and associated supporting functions to encode the student information that is stored at IITH-ACAD. Think of proper data to store the information like the following:

- Names (think of first_name, middle_name, last_name)
- IDs: (in caps or in lowercase)
- email-IDs (can derive it from IDs)
- Program (UG/PG/PhD etc.)
- Dept.
- ….

3.3. [Academic course records as structs] Do a similar exercise for the course records at IITH.

3.4. [Unions] Create a union with various types (int, char, long, array of int). Print the size of the union. Assign values to the members of the union and print their values.

3.5. [Bit fields] Create a struct with bit field members. Print the size of the struct. Compare its size with a struct that declares members of the same type but without bit fields.