Exercises advanced topics

Complete the stringlist demo

1	Implement delete
2	Implement read
3	Use valgrind to find memory errors

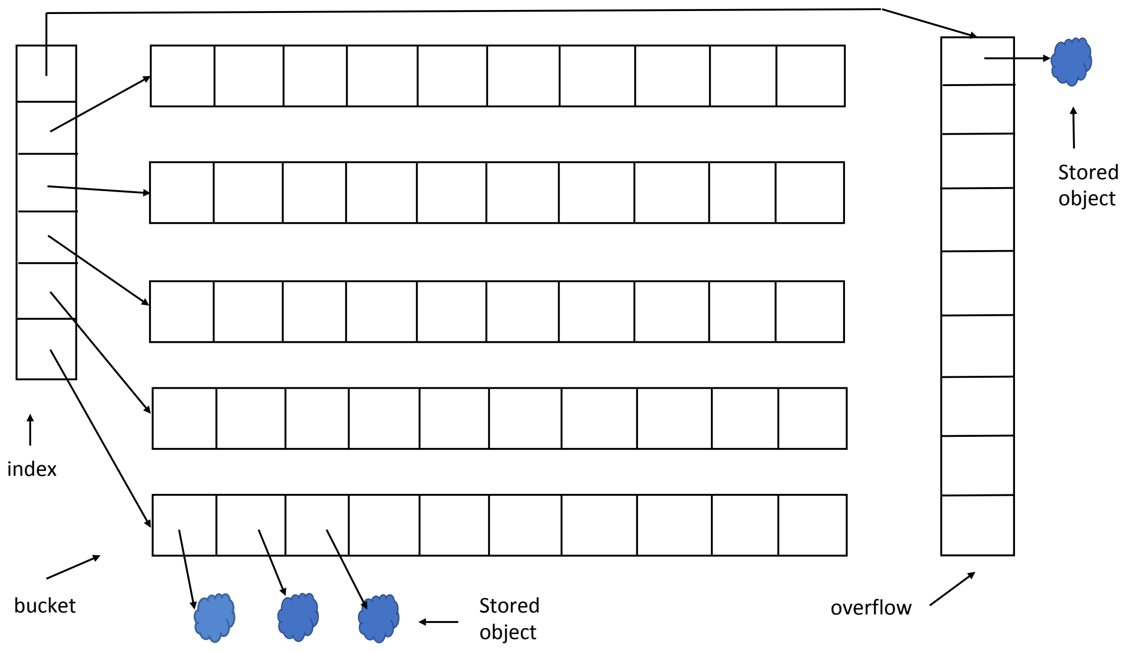
Given the following api called ArrayDataStore

```
void init(int n);
/* pre : n>0
   post: an in-memory data storage for n strings created
*/
void destroy();
/* pre : true
   post: allocated memory de-allocated
*/
bool insert(char * str);
/* pre : str!=null and contains a string STR (\0 as sentinel), not
         necessary dynamically allocated, storage exists
   post: string STR stored in an available free storage slot and 1
         returned, or 0 returned if there are no free storage slots.
*/
void delete(char * str):
/* pre : str!=null and contains a string STR (\0 as sentinel), not
         necessary dynamically allocated, storage exists.
   post: each stored string which matches STR removed, and storage
         slot set free
*/
void print();
/* pre : storage exists
   post: stored strings printed, one string per line
*/
```

Constraints
- Use a char array of strings as internal data structure
- Use global variables to store the data.
- Memory allocation via malloc() and free()
- Structured programming by using header/source files
- Be sure your solution exactly matches the pre/post requirements

4	Implement this api and write a main with some tests
5	Create a Makefile of type 3 (see slides)
6	Write unit tests using Check

Simple Indexed file structure



Constraints
- Use of of structures
- Memory allocation via malloc() and free()
- Building by Makefile, so command make has to be used
- Structured programming by using header files
- Test via unit tests

Tips and specs:
- Create at start up the internal data structure with fixed sizes for index, buckets and overflow.
- "Stored objects" can have different sizes but have the same data elements, like: id, name, address: {12, "Your name", "Address"} has size 4 +(9+1)+(7+1) = 22 and {234, "My Name", "My address"} has size 4+(7+1)+(10+1) =23
- Use hashing to calculate the number in the index array
- Search for the first free position in the assigned bucket
- In case there is no free position in this bucket than search for the first free position in the overflow
- In either case store the "stored object" at the free bucket position.

7	Define the necessary structures
8	Write the necessary functions which manages (allocate and deallocate) the internal data structures.
9	Only implement the CRD of the CRUD operations.
10	Persist the in memory data in a JSON formatted text file.
11	Write unit tests using Check

Client/server sockets, pointer to functions, multi dimensional arrays.

Study the examples client.c and server.c in https://www.geeksforgeeks.org/socket-programming-cc/

Answer the following questions:

12	Which data members has `struct sockaddr_in`, replace the `memset()` function by an explicit initialisation of the other data members, for two of them that's already done:
	
	`serv_addr.sin_family = AF_INET;`
	`serv_addr.sin_port = htons(PORT);`

13	Find out what the possible values of the setsockopt() function like: `SOL_SOCKET, SO_REUSEADDR, SO_REUSEPORT` mean.
14	Use the 12connect wlan the connect your client with the server of an other student. Implement a simple client/server chat program. You can work together in pairs, one implements the client the other one the server.
15	Rewrite example code func-pointer-array.c in such a way that the `funtptr` array gets dynamically allocated memory.
16	**High degree of difficulty** (includes sockets, threads and mutexes) Implement a multiple client version of your chat program and use mutexes to guard the chat-room data structure.

Intro Runtime Mapping

Modern (embedded) microprocessors are equipped with multiple different processor cores. For example your Intel or AMD processor has probably multiple equivalent cores (multi-core) but also dedicated processor core(s) optimised for special operations. The benefits are among others: performance and power consumption.
Run time mapping, a run time service feature in modern Operating or Run time support systems deals with managing scarce resources (like battery, timing) problems during run time.
The following 2 exercises will introduce you to this service by first confront you with a special parallel vector processor already on board of your many-core processor in your laptop. The second exercise shows you how to select the most relevant (functional equivalent) algorithms that your run time environment uses to optimise consumption of relevant resources.

17	**Coding inner product on the AVX vector processor** The AVX vector processor is a 256 bit wide processor that can for example process 8 single point precision floats in a single clock cycle (so 8 x 32 bit floats == 256 bit). Details can be found in: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX2M. The following steps should be followed: 1. Create 2 float arrays on the heap: • Each of size 1M, with values 0.0, 1.0, 2.0 …, 1M-1 for the first vector array (1M==1024^2). • The 2nd vector array contain values 1M, 1M+1…, 2M-1. • These 2 arrays should be placed on a 32 byte boundary on the heap

	2. Create a function that computes the inner product on the CPU, both with single (float) and one with double precision (double) 3. And now on the AVX for both single as well as double precision 4. Compute the *relative error* of these 4 results (correct outcome should be: 960766820994252800) 5. Create a macro that performs an aligned_alloc(boundary, size) as mentioned in Section 4.2 and test it.
18	**Pareto optimal algorithms, making up Pareto diagrams** A modern run time system manages a list of relevant as well as competitive algorithms for each of the functionalities of a program. This list of algorithms is a result of a Pareto analysis that reduces the number of relevant algorithms to a minimum. After this analysis has been completed at design time, the run time system can then automatically select the best algorithm in order to optimise the system's behaviour. The following steps should be followed: 1. Extend the wk6 inner product exercise with time measurement and collect results in an Excel spread sheet (T column), check *pareto.xlsx* 2. Extend this work with power measurement (P column) 3. Copy in the relative errors from wk6's exercise (Q^{-1} column) 4. Use Excel's Scatter diagram method to draw Pareto points in following 3 graphs: ▪ $Q^{-1} \leftrightarrow T$ ▪ $Q^{-1} \leftrightarrow P$ ▪ $P \leftrightarrow T$ 5. Draw the Pareto frontier. What conclusions can you draw from the 3 graphs for the relevancy of algorithms?