Task 3 – Base Digital Objects in Binary File

Develop a program that implements a database of "digital objects" through a binary file. A digital object is anything that can be represented as an array of bytes, while the database can contain many such objects. The desired functionality is described below. The exact format
of the messages printed by the program is described in a separate section.

Function description

The program accepts and executes the following commands repeatedly:

- **o(open) <dbname>:** Open and check database file. The user gives the name of the file dbname in which the database is (will) be stored. If a base file is already open, it is closed before the new one is opened. If the file does not exist, it is created and initialized appropriately. If it exists but is not properly structured (no expected type meta-info - see below), then an error message is printed. If the file cannot be opened/created, then an error message is printed.

- **i(mport) <fname> <objname>:** Import object. The user gives the name of the file fname that contains the object he wants to import into the database, and the name objname with which he wants to save the object in the database. If the file fname provided by the user does not exist, an error message is printed. If an object with the same name name already exists in the base, then an error message is printed.

- **f(ind) <objname>:** Search for objects. The user gives the objname of the object they are looking for. The program searches for all objects in the database with names that contain (anywhere in them) the name given by the user, and prints their full names. If the user enters **"*"** then the program prints the names of all objects present in the database.

- **e(xport) <objname> <fname>:** Export object. The user gives the name of the object objname to export and the name of the file fname in which to save it. The program locates the object in the database and copies its contents to the file specified by the user. If the object does not exist in the base, an error message is printed. If the file specified by the user does not exist, it is created by the program. If the file cannot be created or already exists, an error message is printed. In any case, the object remains in the base.

- **d(elete) <objname>:** Delete object. The user gives the name of the object he wants to delete from the database. The program finds the object, deletes it, and the base file is truncated accordingly. If the object does not exist, an error message is printed

- **c(lose):** Close base file.

- **q(uit):** Quit. The program terminates after first closing the base file (if the user has not already closed the base file via close).

Program entry/exit

A newline character is printed before and after each output message.

Mode	Entrance	output stdout (success)	output stdout (failure)
open	o <dbname>		***Error opening <dbname>.*** the ***Invalid db file <dbname>.***
import	i <fname> <name>		***File <fname> not found.*** the ***Object <name> already in db.*** the ***No open db file.***
find	f <name>	## ***<object name>*** ... ***<object name>***	***No open db file.***
export	e <name> <fname>		***Object <name> not in db.*** the ***Cannot open file <fname>.*** the ***No open db file.***
delete	d <name>		***Object <name> not in db.*** the ***No open db file.***
close	c		***No open db file.***
quit	q		

Implementation requirements

- Define the base functions interface via a header file **(objdb.h)** that contains the definitions for data types that you may have defined as well as the function prototypes for the base functions (initialize a new base file by adding type information, check type information for existing base file, search, import, export, delete objects, close base file).

- In a separate file **(objdb.c),** implement the functions defined in the header file. Functions must not read from conventional input, nor print to conventional output. For the search function in particular, think about how it will return the results to the main program so that it can print them whenever it needs to. Functions that implement import, export, and delete operations must (internally) use the lookup function. When inserting an object into the base, the size of the object must be stored as an int (binary), not a string.

- Develop a control program **(project3.c),** which allows the user to create new or open existing databases and perform the above operations on them by giving appropriate commands.

- Write a suitable **Makefile.** The two .c files must be compiled separately. The final executable should be called project3 and be produced by the first target.

- You can assume that the maximum size of user-supplied dbname, fname, and name is 255 bytes (without the terminal character). The names dbname, fname may also contain path information.

- Files must be accessed using stdio library functions. Your program must check the return value of each call (as well as via the ferror function) to see if the requested operation was executed successfully. If an error occurs that the program itself cannot handle so that its execution can continue without interruption, the program must terminate, after printing an appropriate error message to stderr.

- Meta-information must be inserted at selected points in the database file to check if the file has the expected/correct format/structure (more on this in the tutorial).

- Searching for objects in the database must be done by reading the contents of the database file each time. No "index" information may be stored in program memory.

- The transfer of the contents of the objects from/to the base must be done in parts through a table of size 512 bytes. The memory for this can be allocated by the program statically or dynamically (whatever you think is best).

- Truncation of the base file when deleting objects must, exceptionally, be done using the truncate system function (recall that the stdio library does not provide a function to reduce the size of files). **Caution:** Before calling truncate you must first close the base file with fclose. After calling truncate you can open the file again with fopen.

- There must be at most one open stream in the database file at any time. In other words, you are not allowed to call fopen multiple/consecutive times for the file

of the base, but you are allowed to call fopen to create a new stream in the base file after closing the previous stream with fclose.

- The message “No open db file.” is printed if the user tries to run one of the relevant functions without first successfully opening a base.
- It is forbidden to use a helper file to implement database functions.
- Global/static variables and the use of goto and gets are prohibited.
- Give descriptive names to functions and variables, comment out functions and any parts of the code that are unclear, and write legibly with proper alignment. Code that is "unreadable" is rejected without review.

Code development recommendations

1. Design the organization of data in the database file so that import, search, export, and deletion processes are supported with relative ease. Use paper and pencil to draw the organization you have in mind. Do not start writing code until you are sure that the file organization scheme supports all the needs of the program.

2. Develop the basic functions in **objdb.c** and the corresponding program functions, in a structured way and in the following order: (a) initialize a new base file with type information, (b) check type information in an existing base file, (c) import, (d) search, (e) export, (f) delete. Whenever you complete the implementation of a new function, test it (within the main program that will call the function when the user requests it) to make sure the implementation is correct, before you move on to implementing the next function/function of the program . Test with "borderline" cases, e.g., insert into empty database, search in empty database, search for object not in database, search for object at start/end of file, delete from start/end of database, etc .

3. In addition to the basic functions that you will implement in objdb.c for which you must create separate functions, try to create functions for all parts of the code that are repeated in more places, as well as for individual functions that you want to control independently .

Notes • Pay special attention to the requirement of how many open streams there must be at any time in the file.

- Make sure your program does not try to perform import, export operations etc. on an unopened basis.
- If autolab shows you an error message “UnicodeDecodeError: 'utf-8' codec can't decode byte” then your program is printing to the conventional output data that does not

is ascii (probably during the find process and probably because you made a mistake in "counting" the bytes).

- **ALWAYS CHECK WHAT fread, fseek, etc. RETURN.**

For those who are in the mood

- Extend your implementation so that entire subdirectories (not just individual files) can be imported and exported. Specifically, if the name given to the import is a directory, then all (normal) files contained in the directory (without any special files or other subdirectories) should be imported, properly associated with the name of the directory. Accordingly, if the name given to export/delete is the name of a directory then the directory with all files belonging to it should be exported to the local directory / deleted from the database. Finally, find should also work for files inside a directory entered in the base, in which case the full name/path will be printed. To implement this functionality you will need to use the opendir, readdir and mkdir functions. Carefully read the corresponding man pages on your system.

- For even more ideas, contact the teachers.

Packing and shipping work

1. Create a directory named project3submit
2. Copy project3.c, objdb.h, objdb.c, Makefile to directory project3submit
3. Pack and compress the project3submit directory by right-clicking and selecting Compress here as .tar.gz
4. Login to autolab, and select project3 of the ECE116-S20 course
5. Accept the academic integrity message, then click SUBMIT.
6. In the window that will appear, locate and select the project3submit.tar.gz file you made to upload it to autolab.
7. After one minute, refresh the page to see your grade on the individual tests.