

**Array of structured variables:****Simple Water System Database**

For this project, you will implement a simple database system for account information in a simple water system. Your program will create an in-memory database using an array of `struct` variables representing account records, and then perform some common operations database on the array of account records. This project will make use of almost everything we have covered in the course, including arrays and `struct` types, searching and sorting, and script-driven execution.

The database will be initialized by reading account records from an input file and storing the data in a single array, using an appropriate `struct` type that you must design for this assignment. As soon as the initial database has been created, it must be displayed to a log file. Then the program will process commands from a second input file, which will entail performing some operations on the database, and logging results. After the last command has been processed, the final database must be displayed to the log file, and the program must terminate.

This is, by far, the most complex project in this course. In order to succeed, you must take the design process seriously, and you must practice incremental development. Since that has been discussed at some length in other project specifications, no guidelines for step-by-step implementation or testing are provided, but you certainly should plan to implement and test your implementation one step at a time.

The comments and requirements about representing monetary amounts that were made in the specifications for the previous projects still apply.

**Rate structure:**

The amount due is calculated from the number of gallons used, but the exact rules depend on whether the customer is residential (code R) or commercial (code C).

For residential customers, the amount due is based upon the following rules:

- The base rate fee of \$22.50 is charged regardless of the water usage.
- If the water usage exceeds 1000 gallons, then an additional fee of \$0.10 per gallon is charged for the gallons in excess of 1000.
- If the water usage exceeds 1500 gallons, a surcharge of \$10.00 is also applied.

For commercial customers, the amount due is based upon the following rules:

- The base rate fee of \$40.00 is charged regardless of the water usage.
- If the water usage exceeds 2000 gallons, then an additional fee of \$0.08 per gallon is charged for the gallons in excess of 2000.
- If the water usage exceeds 4000 gallons, then a surcharge of \$0.05 per gallon is also applied for the gallons in excess of 4000.

For example, a residential customer using 1738 gallons of water would be charged  $\$22.50 + \$0.10 * 738 + \$10.00$  or \$106.30.

**Database script commands:**

Each command will begin with a command phrase, followed immediately by a single tab character, and then by one or more whitespace-separated integer values, depending upon the particular command word. The commands file will be syntactically correct, but the commands may entail logical errors such as specifying non-existent account numbers, and your design must deal with those sensibly.

Here is a description of each of the commands your program must recognize and process:

**find**<tab><account number>

Search the database for a record that matches the given account number. If a match is found, log the index at which the record is located in the array, the account number, the rate code, and the gallons used. If no match is found, log the account number followed by the error message "not found".

**add**<tab><account number><ws><rate code><ws><gallons used>

If the database array isn't full, append a new record to the database, representing the given account data and log the account information, just as for the `find` command. If the database is full, log the error message "database full".

**delete**<tab><account number>

Search the database for a record that matches the given account number. If a match is found, remove that record from the array, and log the message "Removed" followed by the index at which the matching record was found. If no match is found, log the account number followed by the error message "not found".

**sortby**<tab>[ **AccountNumber** | **RateCode** ]

Sort the database so that the records are listed in ascending order, according to the field specified in the command. Log the message "Swaps" followed by the number of swaps performed by the sort algorithm. Note: you are required to use the bubble sort algorithm for this; if you use another sorting algorithm, such as selection sort, you will report the wrong number of swaps.

**display**<tab>

Log the contents of the database, as shown in the log file given below. If the database is empty, log the message "no records".

**exit**<tab>

Immediately stop processing the script file, and log the message "Finished script".

### Sample input and corresponding output:

The initial account data will be supplied in a file named "AccountData.txt". Each line specifies the account number, rate code, and gallons of water used for a different account. The values are separated by one or more whitespace characters, not necessarily including tabs.

W32954	C	2049
W33208	R	2145
W33081	R	1407
W33032	R	1262
W33131	C	2179
W33301	C	1522
W33378	C	967
W33083	R	1228
W33441	C	2710
W33164	C	1646
W33508	R	1738
W33574	C	665
W33634	C	2991
W32902	C	1824

The initial account data file is guaranteed to conform to this description. The account numbers will be strings, not containing any whitespace characters. The rate code will be a single character, either 'R' or 'C'. The number of gallons used will be a nonnegative integer. There will never be data for more than 50 accounts.

The database commands will be supplied in a file named "Script.txt". Each line of the script file will either be a comment or one of the commands described previously.

```
; Water system test script
;
; Look up some accounts:
;
find      W32902
find      W33208
find      W33634
; Add some new accounts:
;
add       W03608    R    2487
add       W05664    C    2133
add       W10375    C    1311
display
; Delete some accounts:
;
delete    W32902
delete    W33208
delete    W33634
display
find      W32902
find      W33208
; Sort the account list
;
sortby    AccountNumber
find      W32954
find      W33574
; Sort the account list
;
sortby    RateCode
find      W32954
find      W33574
; Quit
exit
```

Each command is guaranteed to conform to the syntax given earlier. There is no limit on the number of comments, or commands, that may be given in the script file, not that it should matter to your implementation. Each script file will contain an `exit` command.

Here is a sample output file for the program, which must be named "AccountLedger.txt".

Programmer: Bill McQuain  
CS 1044: Water System Ledger

	Acct #	Gal Used	Amt Due	Code
0:	W32954	2049	43.92	C
1:	W33208	2145	147.00	R
2:	W33081	1407	63.20	R
3:	W33032	1262	48.70	R
4:	W33131	2179	54.32	C
5:	W33301	1522	40.00	C
6:	W33378	967	40.00	C
7:	W33083	1228	45.30	R
8:	W33441	2710	96.80	C
9:	W33164	1646	40.00	C
10:	W33508	1738	106.30	R
11:	W33574	665	40.00	C
12:	W33634	2991	119.28	C
13:	W32902	1824	40.00	C
-----				
find	W32902			
13	W32902	1824	40.00	C
-----				
find	W33208			
1	W33208	2145	147.00	R
-----				
find	W33634			
12	W33634	2991	119.28	C
-----				
add	W03608	2487	R	
14	W03608	2487	181.20	R
-----				
add	W05664	2133	C	
15	W05664	2133	50.64	C
-----				
add	W10375	1311	C	
16	W10375	1311	40.00	C
-----				
display				
	Acct #	Gal Used	Amt Due	Code
0:	W32954	2049	43.92	C
1:	W33208	2145	147.00	R
2:	W33081	1407	63.20	R
3:	W33032	1262	48.70	R
4:	W33131	2179	54.32	C
5:	W33301	1522	40.00	C
6:	W33378	967	40.00	C
7:	W33083	1228	45.30	R
8:	W33441	2710	96.80	C
9:	W33164	1646	40.00	C
10:	W33508	1738	106.30	R
11:	W33574	665	40.00	C
12:	W33634	2991	119.28	C
13:	W32902	1824	40.00	C
14:	W03608	2487	181.20	R
15:	W05664	2133	50.64	C
16:	W10375	1311	40.00	C
-----				
delete	W32902			

Removed 13

delete W33208  
Removed 1

delete W33634  
Removed 11

display

	Acct #	Gal Used	Amt Due	Code
0:	W32954	2049	43.92	C
1:	W33081	1407	63.20	R
2:	W33032	1262	48.70	R
3:	W33131	2179	54.32	C
4:	W33301	1522	40.00	C
5:	W33378	967	40.00	C
6:	W33083	1228	45.30	R
7:	W33441	2710	96.80	C
8:	W33164	1646	40.00	C
9:	W33508	1738	106.30	R
10:	W33574	665	40.00	C
11:	W03608	2487	181.20	R
12:	W05664	2133	50.64	C
13:	W10375	1311	40.00	C

find W32902  
W32902 not found

find W33208  
W33208 not found

sortby AccountNumber  
Swaps: 40

find W32954  
3 W32954 2049 43.92 C

find W33574  
13 W33574 665 40.00 C

sortby RateCode  
Swaps: 28

find W32954  
2 W32954 2049 43.92 C

find W33574  
8 W33574 665 40.00 C

exit  
Finished script

	Acct #	Gal Used	Amt Due	Code
0:	W05664	2133	50.64	C
1:	W10375	1311	40.00	C
2:	W32954	2049	43.92	C
3:	W33131	2179	54.32	C
4:	W33164	1646	40.00	C
5:	W33301	1522	40.00	C
6:	W33378	967	40.00	C

---

7:	W33441	2710	96.80	C
8:	W33574	665	40.00	C
9:	W03608	2487	181.20	R
10:	W33032	1262	48.70	R
11:	W33081	1407	63.20	R
12:	W33083	1228	45.30	R
13:	W33508	1738	106.30	R

As before, it begins with two lines identifying the programmer (you) and the specific project, followed by a blank line. That is followed by a display of the initial database, then by the results from processing the commands, and finally by a display of the final database.

If you have read the *Student Guide to the Curator*, you already know that all the fixed text must be precisely as shown in the sample output. The output should be aligned for easy readability.

Additional samples of input and correct output will be available on the course website.

### Suggested implementation plan:

As always, you should design your solution piece by piece. However, you can certainly take advantage of the design work you did for the previous water bill project, not to mention the chance to reuse some of the actual code from that project. Begin by identifying the major tasks that have to be done, and then add detail for each of those tasks. Your implementation should be developed in the same manner.

Given the complexity of this program, it is imperative that you manage your implementation sensibly. The obvious starting point is to read the initial account data into an array of `struct` variables, and echo it to the log file. From there, you should add support for each specified database command, and test as you go.

You should make sure that your program will simply skip over any commands that it does not recognize. That will make it easier for you to debug it, and also give you a better chance of getting a decent score even if you don't manage to get all of the specified features working.

### Evaluation:

Everything that you have been told about testing in class applies here. Do not waste submissions to the Curator in testing your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that are provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 100, based upon the runtime testing performed by the Curator System. We will also evaluate your submission of this program for documentation style and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1044 website for general guidelines. You should comment your code in the same manner as the code given for the first two programming assignments. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every function aside from `main()` must have a header comment block as described on the course website.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Adopt a consistent indentation style and stick to it.

Your implementation must also meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ string variables to hold character data, not C-style character pointers or arrays.
- Note: you are explicitly required to write user-defined functions for this program. In particular, you are required to implement at least ten functions besides `main()`. (Mine uses 18.) You should make appropriate use of `void` and non-`void` functions.
- Pass parameters appropriately. Do not use pass-by-reference unless it is logically necessary, especially with parameters that are arrays.
- Do not declare any global variables! There will be a substantial penalty if you do. Global declarations of functions, and constants, and of the `struct` type are OK.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

### Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* can be found at: <http://www.cs.vt.edu/curator/>

The submission client can be found at: <http://eags.cs.vt.edu:8080/curator/>

### Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
```

**Failure to include this pledge in a submission is a violation of the Honor Code.**