

Parallel Arrays for Data Organization:**Extended Investment Tracking**

For this project, you will extend the simple C++ program you wrote for project 4. All of the calculations that were specified for that assignment are retained, and the input file format will be the same. There are two primary differences. First, your implementation now must reflect a good procedural decomposition of the problem. Second, the table of account information must be written in sorted order. That means that you must read and store all of the account data in memory, at once, rather than processing line-by-line; and that means that you must use a collection of "parallel" arrays to organize the account data.

You will be given the same data about each investment account as before, including the account name, current balance, rate of return, and a brief description of the account. As before, you will compute the change in the balance and the new balance for each account, and a summary that specifies the total of the initial balances, the total increase in value according to the given rates of return, the total of the final balances of the accounts, and the overall rate of return.

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

Sample input and corresponding output:

Here is a sample input file for the program, named "AccountInfo.txt". The first two lines form the top of a table of data. Each of the remaining lines specifies the name, current balance, rate of return, and a description of an investment account.

Account	Balance	Rate	Description
WFCD	7277.37	0.006	24 month CD
Dreyfus	18966.40	-0.069	money market account
Savings	2664.05	-0.054	savings account
Checking	706.16	-0.009	checking account

The input file is guaranteed to conform to this description. The balances will always be nonnegative decimal numbers, but the rates of return may be negative. The account names will never contain spaces or tabs (which does have something to do with how you will read them). The descriptions may contain any characters, including whitespace, and there is no given bound on how long the descriptions may be. The data values are separated by spaces, not by tabs.

There will never be data for more than 50 investment accounts; your solution must work with any number of accounts up to that limit.

Here is a sample output file for the program, which must be named "AccountReport.txt". As before, it begins with two lines identifying the programmer (you) and the specific project, followed by a blank line.

There are two lines forming the header for a table of account information, as shown. Each line of the table will report the account name, initial balance, change, and final balance. In addition, if there was a loss of \$1000.00 or more, the message "ouch!" will be printed, and if there was a loss of \$100.00 or more but less than \$1000.00 the message "oops!" will be printed as shown. The lines of account data must be sorted so that the amounts in the change column are in decreasing order.

After all the accounts have been reported, there is another delimiter line to mark the bottom of the table, followed by a line that reports the totals for each of the table columns.

That is followed by a blank line, and then a line that reports the overall rate of return, which must be reported with precision 3 as shown. 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.

```

Programmer:  Bill McQuain
CS 1044:    Account List

```

Account Name	Initial Balance	Change	Final Balance	
WFCD	7277.37	43.66	7321.03	
Checking	706.16	-6.35	699.81	
Savings	2664.05	-143.85	2520.20	oops!
Dreyfus	18966.40	-1308.68	17657.72	ouch!
Totals	29613.98	-1415.22	28198.76	
Yield:	-0.050			

Note: when printing negative monetary amounts, you may encounter some difficulties because you are storing them using integer values. If you do, you might find one of the mathematical functions built into the Standard C++ libraries useful. There a function to take the absolute value of an integer, `abs()`, that is declared in the header file `cmath`.

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 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.

The most significant change is that this program must employ the input-failure control logic covered in class; be sure you understand the examples in the notes and that you follow the correct design pattern.

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 may 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 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 must employ at least 5 functions besides `main()`.
- You must use the selection sort algorithm to sort the account data.

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.