

## Structure Array, Character Strings, Searching and Sorting

For this assignment, there are requirements that you may be graded upon in addition to producing the correct output. They are: (1) you must write an outline of your program design that reflects your top-down functional/modular decomposition – this must be included in your comments at the beginning of your program, (2) your program should contain good documentation, (3) you must write and use an array of records, implementing your design as functions in your code. The GTAs may HAND-CHECK your code on this assignment to grade you on these items. You should be sure to read and apply the documentation standards as referenced in the Programming Standards section of this specification. This programming assignment uses many of the ideas presented in sections 10 and 11 of the course notes. You are advised to read those carefully. Read and follow this program specification closely.

### The Program Specification:

### Movie Rentals

A local movie rental company needs to track its tapes. The program will read a file that contains records with data about the movies in the store, (see input file description). An appropriate structure type must be declared for use in an array of records to store all of the movie information.

Your program will first input all of the movie records. For simplicity, you may assume that there will be no more than 100 movie records. (You should code your program using a constant for this value to allow it to be easily modifiable.) The program will then produce a listing of the movie data sorted by movie titles. Following the sorted listing, the program will determine and output two sets of statistics. The first set will give counts of how many movies in the input file are at what Rating level (G, PG, PG13, R, X). The second set will give counts of the review levels. Reviewed movies are given a star count that signifies how good the critic judged the movie to be. The star counts range from [0.0 , 5.0] with the limitation that the decimal portion of the review can only be zero or one-half.

### Input file descriptions and samples:

Your program **must** read its input from a file named `movies.dat` — use of any other file name will result in a score of zero. The `movies.dat` file will store the movie record information, one record per every two lines in the file. The format of the `movies.dat` file will now be described. The first line of the input file is a label line that must be ignored. Starting on the second line each movie record begins with six data fields, separated by one tab character. The order of the data fields on the line and the type of value in the field are given in the table at the right. Following on the second line of each movie record will be a character string that contains a brief synopsis/review of the movie.

Movie Data Field Name	Field Contents
Title	character string
Director	character string
Release Year	integer
Performers	character string
Rating	character string
Stars	real

A sample `movies.dat` input file is shown below, (the first two lines are column labels and are not part of the file):

```
0000000011111111112222222223333333334444444445555555556666666667777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
// Title      Director      Release Performers      Rating Stars
Dragon Slayer  Unknown      0      R Richardson, P McNichol  G      3.5
Whoever produced this movie should be Walt Disney's son.
Alien         Cameron      1978  Sigourney Weaver      PG13  4.0
The special effects were not only technically good, but innovative.
Star Wars    G Lucas      1976  Hamill, Ford, Fisher   G      5.0
A classic SciFi version of the eternal conflict of Good verses Evil.
Blade Runner R Scott      1982  H Ford, R Hauer       R      3.0
This movie is a cult classic, a must see.
Predator     Unknown      1988  Arnie                  R      3.0
Alien picture where IT doesn't land in boondocks fighting Jeb and Billy Bob.
Terminator   Unknown      0      Arnie, Linda Hamilton  R      2.5
Robot from future trying to change the future.
```

It may **NOT** be assumed that the movie records in the file will be in any particular order. You may however assume that the field data has been entered correctly and is separated by tab characters, (i.e., assume that the data is valid and does not need to be checked for errors). Your program must be written so that it will detect when it's out of input and terminate correctly, just as in the previous projects.

### Output description and sample:

The first line of your output must include your name only; the second line must contain the title "Movie Data"; the third line must be a line of underscore characters; the fourth line must display the column labels shown below. The fifth line will contain movie data from the input file, (omitting the one line reviews), sorted by the movie title, aligned under the appropriate headers. The line following the last record of the title sort must display a line of underscore characters; followed by a blank line. The next line must display column labels for the movie rating categories. Aligned under these labels are the counts of the number of each type of movie in the input file. The line following the rating category counts line must be a line of underscore characters; followed by another blank line. The next line must display column labels for the movie star counts categories. Aligned under the star counts labels are the counts of the number of movies in the input file for each star count numerical rating. The last line of output following the star counts line must be another line of underscore characters.

Your program must write output data to a file named `film.dat` — use of any other output file name will result in a score of zero. The sample output file shown below corresponds to the input data given above, (the first two lines of column labels are not part of the file):

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Dwight Barnette

Movie Data

Title	Director	Release	Performers	Rating	Stars
Alien	Cameron	1978	Sigourney Weaver	PG13	4.0
Blade Runner	R Scott	1982	H Ford, R Hauer	R	3.0
Dragon Slayer	Unknown	0	R Richardson, P McNichol	G	3.5
Predator	Unknown	1988	Arnie	R	3.0
Star Wars	G Lucas	1976	Hamill, Ford, Fisher	G	5.0
Terminator	Unknown	0	Arnie, Linda Hamilton	R	2.5

Ratings:	G	PG	PG13	R	X
	2	0	1	3	0

Stars:	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
	0	0	0	0	0	1	2	1	1	0	1

Note that due to the varying lengths of the character strings alignment of the labels will present a challenge. You are **NOT** required to use this exact horizontal spacing, but your output must satisfy the following requirements:

- You must use variables of type `int` for the counts of the movie ratings and star reviews.
- You must use variables of type `double` for the star review value.
- All decimal values must be printed to show the same number of decimal places, as in this sample.
- You must use the exact specified header and column labels given above.
- Include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with the identifying labels shown in the example. Note that while the auto-grader doesn't deduct points for horizontal alignment, the TA who may grade your source code for programming standards may do so.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line, including the last line.

## Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. A copy of *Elements of Programming Style* is included with the course notes and is available on-line from the course Web site. Some specific requirements follow.

### Documentation:

- For the outline of your design:  
You must include, below the header comments, an outline program design. The design must reflect your top-down functional/modular decomposition of the problem. The design should follow the layout of the payroll design example in appendix 4 of the course notes.
- You must include header comments specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant in your program.
- You must use meaningful identifier names that suggest meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- Each user-defined function must have a valid C++ prototype and the definition of each user-defined function (except `main`) must be preceded by a block comment, explaining in one sentence what the function does, and listing each parameter to the function and explaining its purpose.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable and comment each logical section/structure.
- You may copy, (that's copy not move), parts of your design outline from the top of your file to relevant points in your code to use as comments and to clearly show how your code matches your design. However, unless you have created an very "complete" design, you will still need to supplement it with inline code comments as explained in the "Elements of Programming Style".

### Implementation:

- Use named constants instead of variables or literal constants where appropriate, (e.g., array dimensions).
- Use `bool` variables, `typedefs` and `enum` types where appropriate.
- Do not use `if...else` statements, when a `switch` statement can be used.
- Choose your control structures appropriately. Do not use while loops for count controlled loops.
- Your implementation must minimally include **five** user-defined functions, not counting `main`.
- You may use file-scoped function prototypes, constants and user-defined types.
- You may not use file-scoped variables of any kind.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement other than a constant or variable declaration, function prototype or comment. Blank lines do not count.
- The definition of `main()` must be the first function definition in your source file. The remaining function definitions should be grouped logically.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed.
- Parallel arrays may not be used in this program. Use of an **array of structures** is required.

**Your submission that receives the highest score will be graded for adherence to these requirements, whether it is your last submission or not. If two or more of your submissions are tied for highest, the earliest of those may be selected for grading. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later.**

### Incremental Development:

You'll find that it's easier and faster to produce a working program by practicing incremental development. In other words, don't try to solve the entire problem at once. First, develop your design. When the time comes to implement your design, do it piece by piece. Check your implementation and output to be sure that you've followed all the specifications given for this project, especially those in the Programming Standards section above. At this point, you're ready to submit your solution to the auto-grader.

### Testing:

At minimum, you should ascertain that your program produces the output given above when you use the given input file. However, verifying your program produces correct results on one test case does not constitute a satisfactory testing regimen.

Additional input/output file examples will be posted on the Web site. You may use those for additional testing. You should make up and try additional input files as well; determining by hand what the correct output would be.

### CS 1704:

Students who are anticipating continuing on to the next programming course, CS 1704, in the Fall should save this program. Please be aware that the term project in CS 1704 in the Fall will build off this program. A full-fledged movie rental tracking system will be developed in the next course.

### Pledge:

As with the previous projects, each of your submissions to the Curator 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 automated grader.
```

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