Assignment: Dynamic Memory Allocation and Sorting

Learning Outcomes

- Reading values from files
- Dynamic memory management
- Processing arrays using pointer offsets
- Selection sort algorithm

Task

The objective is to develop a program that analyzes an unknown number of course grades stored in a data file. The program will open the data file, determine the count of grades in the file, allocate appropriate memory in the heap as a container for the grades, store grades read from the file in the previously allocated heap storage, and then analyze the data. The report provides the minimum, maximum, average, and median grades, variance and standard deviation of the grades, and a table indicating the percentage of grades in each letter grade category.

Declare the following functions to solve the problem:

```
double* read_data(char const *file_name, int *ptr_cnt);
double  max(double const *begin, double const *end);
double  min(double const *begin, double const *end);
double  average(double const *begin, double const *end);
double  variance(double const *begin, double const *end);
double  std_dev(double const *begin, double const *end);
double  median(double *base, int size);
void    selection_sort(double *base, int size);
void    ltr_grade_pctg(double const *begin, double const *end,
                       double *ltr_grades);
```

Your submission must satisfy the following criterion:

> ***C standard library function*** `qsort` ***must not be used to sort values. Instead, you must implement function*** `selection_sort` ***to sort values.***

A brief summary of each of the five functions follows:

1. Function `read_data` opens the data file specified by parameter `file_name`, determines the count of grades in the file and writes the count to the location pointed to by parameter `ptr_cnt`, allocates enough heap memory to contain the contents of the data file, and copies the grades from file to this heap memory. If for any reason, the function is unable to complete these tasks, it should return `NULL`; otherwise, the function should return the address of the first byte of the previously allocated heap memory containing the values read from the file.
2. Functions `max`, `min`, `average`, `variance`, and `std_dev` return the maximum, minimum, average, variance, and standard deviation of a half-open range of values.

   The average or *mean* value is computed as:

$$\mu = \frac{\sum_{k=0}^{n-1} x_k}{n}$$

where x represents an array of n values with

$$\sum_{k=0}^{n-1} x_k = x_0 + x_1 + x_2 + \cdots + x_{n-1}.$$

The variance σ^2 for a set of data values stored in an array x can be computed using the following equation:

$$\sigma^2 = \frac{\sum_{k=0}^{n-1} (x_k - \mu)^2}{n-1}$$

The standard deviation σ is defined to be the square root of the variance:

$$\sigma = \sqrt{\sigma^2}$$

3. Function `median` returns the value in the middle of a group of values, assuming that the values are sorted. If there is an odd number of values, the median is the value in the middle; if there is an even number of values, the median is the average of the values in the two middle positions. For example, the median of the values $\{1, 6, 18, 39, 86\}$ is the middle value, or 18; the median of the values $\{1, 6, 18, 39, 86, 91\}$ is the average of the two middle values, or $(18 + 39)/2$, or 28.5. Assume that a group of sorted values are stored in an array and that n contains the number of values in the array. If n is odd, then the subscript of the middle value can be represented by $\lfloor n/2 \rfloor$, as in $\lfloor 5/2 \rfloor$, which is 2. If n is even, then the subscripts of the two middle values can be represented by $\lfloor n/2 \rfloor - 1$ and $\lfloor n/2 \rfloor$, as in $\lfloor 6/2 \rfloor - 1$ and $\lfloor 6/2 \rfloor$, which are 2 and 3, respectively. This function will need a sorted sequence of values and will call the `selection_sort` function (explained below) to sort the values in the array whose first value is pointed to by parameter `base`.

4. Function `selection_sort` will implement the selection sort algorithm to sort `size` number of values of the array whose first value is pointed to by `base` in *ascending order*. There is an important caveat in the implementation of function `selection_sort`:

   > ***Your submission must not use the C standard library function `qsort` to sort values.***

5. Define an enumeration type with enumeration constants in the order `A_GRADE`, `B_GRADE`, `C_GRADE`, `D_GRADE`, `F_GRADE`, and `TOT_GRADE`. The enumeration constants must have default values starting from 0 (for `A_GRADE`) with subsequent constants having a value one larger than the previous constant. Therefore constant `TOT_GRADE` will have value 5

6. Suppose each value in the data file represents the final course grade in the range from 0 to 100. Function `ltr_grade_pctg` writes the percentage of values in the data file that map to a letter grade. A value in the range $[0,100]$ is mapped to a letter grade using the table on the last page.

   Enumeration constants `A_GRADE` through `F_GRADE` specify the subscript of the corresponding letter grade A through F. These subscripts are used for the array whose first element is pointed to by parameter `ltr_grades`. Study the use case of this function in the driver to avoid confusion about the purpose and use of this function.

Header and Source Files

As usual, you'll define the enumeration type and declare the functions in `q.h`. Include all header files that you find necessary to implement the corresponding definitions in `q.h`. Define the functions declared in `q.h` in source file `q.c`. You can use Standard C library functions to implement the functions. Don't forget to include `q.h` in `q.c` to successfully compile `q.c`.

Download the driver source file `qdriver.c`, a *makefile*, input files containing course grades, and corresponding (correct) output files. Use the strategy of providing *stub* functions to establish and verify linkage between source files `qdriver.c` and `q.c`, and implementing and verifying the correct behavior of one function at a time. Build executable `q.out` using *makefile*:

```
$ make
```

and use command-line parameters to specify the input and output file names to the program:

```
$ ./q.out grades1.txt myoutput1.txt
```

Compare the output from your implementation with the correct implementation in `output1.txt` using `diff`:

```
$ diff -y --strip-trailing-cr --suppress-common-lines myoutput1.txt
output1.txt
```

Submission and Automatic Evaluation

1. In the course web page, click on *Assignment 10 Submission Page* to submit `q.c` and `q.h`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
   - F grade if your `q.c` doesn't compile with the full suite of `gcc` options.
   - F grade if your `q.c` doesn't link to create an executable.
   - F grade if output of function doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests).
   - F grade if submission uses C standard library function `qsort` or uses the subscript operator.
   - The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission. $A+$ grade if output of function matches correct output of auto grader. The data files `grades?.txt` and corresponding output files `output?.txt` used by the auto grader have been provided to you.
   - A deduction of one letter grade for each missing documentation block in `q.c` and `q.h`.

     Your submission `q.c` must have one file-level documentation block and **nine** function-level documentation blocks. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an $A+$ grade and one documentation blocks are missing, your grade will be later reduced from $A+$ to $B+$. Another example: if the automatic grade gave your submission a C grade and the two documentation blocks are missing, your grade will be later reduced from C to E.

> Likewise, your submission `q.h` must have one file-level documentation block and **nine** function-level documentation blocks. Remember, your clients will not have access to the definitions in `q.c`.

Selection Sort Algorithm

One of the most common applications in computer science is sorting - the process through which data are arranged according to their values. We're surrounded by data. If data were not ordered, we would spend hours trying to find a single piece of information. Imagine the difficulty of finding someone's telephone number in your cellphone's contact list that was not ordered! In this document, we present a sorting algorithm called the selection sort that is simple to understand and straightforward to code in a function.

Although the type of elements in the array is of no significance to the algorithm itself, assume an array whose elements are assigned arbitrary integral values. The selection sort algorithm begins by finding the minimum value and exchanging it with the value in the first position in the array. Then the algorithm finds the minimum value beginning with the second element, and it exchanges this minimum with the second element. This process continues until reaching the next-to-last element, which is compared with the last element; the values are exchanged if they are out of order. At this point, the entire array of values will be in ascending order. This process is illustrated in the following sequences:

Original order:

5	3	12	8	1	9

Exchange minimum with value in subscript 0:

1	3	12	8	5	9

Exchange next minimum with value in subscript 1:

1	3	12	8	5	9

Exchange next minimum with value in subscript 2:

1	3	5	8	12	9

Exchange next minimum with value in subscript 3:

1	3	5	8	12	9

Exchange next minimum with value in subscript 4:

1	3	5	8	9	12

Array values are now in ascending order:

1	3	5	8	9	12

A hand implementation of the algorithm for a deck of cards can be visualized here. An algorithm for the selection sort can be designed like this:

Algorithm Selection Sort

Input: Array A of n elements
Output: Array A sorted in ascending order

1. $i = 0$
2. while $(i <= n - 2)$
3. find *min:* the index of smallest element in unsorted subarray $A[i]$ through $A[n - 1]$
4. **if** $i \neq min$
5. swap($A[i]$, $A[min]$)
6. $i = i + 1$
7. endwhile

Grade Assignment

Value v	Letter Grade	Subscript
$v \geq 90$	*A*	A_GRADE
$80 \leq v < 90$	*B*	B_GRADE
$70 \leq v < 80$	*C*	C_GRADE
$60 \leq v < 70$	*D*	D_GRADE
$v < 60$	*F*	F_GRADE