Exercise Problems - 13

Structures

1. In this exercise, we will re-do Experiment 1 of Lab-exercise, Week 13, using functions. Download the file data02.txt to your working directory. You will be writing the output data to a file named grades02.txt. Define a structure student with the following fields:

```
char name[15];
int marks[4];
char grades[4];
```

   Assume that `marks[0]` represents marks obtained for Physics, `marks[1]` - marks obtained for Chemistry, `marks[2]` - marks obtained for Mathematics and `marks[3]` - marks obtained for Biology. Similarly, assume that `grades[0]` represents the grade for Physics, `grades[1]` - the grade for Chemistry etc.

   In your main function, define an array `struct student l[200]` for holding the details of all students together, an array `averages[4]` that can hold the average marks of each subject and two file pointers, one each for input and output files. The file opening and closing are to be performed within the `main()` function. Define other variables as necessary.

   Write a `get_input()` function which takes a file pointer `fp` and an array of students `slist` as parameter and reads in the details of all students from the file pointed to by `fp` and update the details of each student in the array `slist`. The function `get_input()` should return the number of students as its return value. If the number of students exceeds 200 while reading from the file, the function should return -1 as an error code, to prevent segmentation fault.

   Write another function `calculate_subj_average()` that takes an array of students `slist`, the number of students `k` and an integer `subject_index` (which is between 0 and 3) as parameters and returns the average marks of students in the subject indicated by the index `subject_index`.

   Write a third function `calculate_subj_grade()` that takes an array of students `slist`, the number of students `k`, an integer `subject_index`, and a float value `subject_avg` as parameters and update the grades of the subject indicated by `subject_index`, using the `subject_avg`. This function should have no return values.

   Write a fourth function `generate_output()` function which takes a file pointer `fp`, an array of students `slist`, and the number of students k as parameters and writes the names and grades of all students into a file pointed to by `fp`.

   Use these functions to design your main function and complete the program.

2. In this exercise, we will calculate CGPAs of all students using the file `grades02.txt` obtained from the previous program and generate a rank list, in the decreasing order of CGPAs and write it to a file named `ranklist.txt`. Define a structure student with the following fields:

   ```
   char name[15];
   char grades[4];
   float cgpa;
   ```

   In your main function, define an array `struct student l[200]` for holding the details of all students together and two file pointers, one each for input and output files. The file opening and closing are to be performed within the `main()` function. Define other variables as necessary.

   Write a `get_input()` function which takes a file pointer `fp` and an array of students `slist` as parameter and reads in the details of all students from the file pointed to by `fp` and update the details of each student in the array `slist`. The function `get_input()` should return the number of students as its return value. If the number of students exceeds 200 while reading from the file, the function should return -1 as an error code, to prevent segmentation fault.

   Write another function `compute_cgpa()` that takes an array of four letter grades as parameters and outputs the cgpa based on the grades. (If s is a student, you will be able to call this function with the four character array s.grades as parameter function to obtain s.cgpa).

   Write a `get_max_posn()` function which takes an array of students `slist` and the number of students `k` as parameters and returns the array index of the student with the maximum CGPA.

   Write a `swap()` function for exchanging the details of two students.

   Write a function `sort_stud_list()` that takes an array of students `slist` and the number of students `k` as parameters and sorts the array in the descending order of CGPAs, by calling functions `get_max_posn()` and `swap()` repeatedly.

   Finally, write a function `generate_output()` function which takes a file pointer `fp`, an array of students `slist`, and the number of students k as parameters and writes the names and CGPAs of all students into a file pointed to by `fp` as per the ranklist.

   Use these functions to design your main function and complete the program.