Institute for Advancing Intelligence, TCG CREST
(TCG Centres for Research and Education in Science and Technology)

Introduction to Programming and Data Structures
Ph.D. Coursework: First year, First Semester (Session: 2024-25)

Assignment #02

Full Marks: 200 — Instructor: Dr. Laltu Sardar

Clarification Deadline: **2024-Sep-03** — Submission Deadline: **2024-Sep-08**

Instructions

Use dynamic memory allocation where necessary, and ensure all dynamically allocated memory is freed appropriately.

Problem #AP0201: Binary to Integer Conversion

- **Function:** `int bin_to_int(const char* binary_string)`
- **Description:** Implement a function `bin_to_int()` that reads a binary number as a string from the terminal and converts it into an integer value.
- **Input:** A string `binary_string` containing a binary number.
- **Output:** The integer value corresponding to the binary number.
- **Memory Management:** No dynamic memory allocation is required.
- **Sample Input:**

```
"1010"
```

- **Sample Output:**

```
10
```

[15]

Problem #AP0202: Binary to Floating-Point Conversion

- **Function:** `float bin_to_float(const char* binary_string)`
- **Description:** Implement a function `bin_to_float()` that reads a binary number, possibly containing a decimal point, as a string from the terminal and converts it into a floating-point value.
- **Input:** A string `binary_string` containing a binary number.
- **Output:** The floating-point value corresponding to the binary number.
- **Sample Input:**

```
"1010.101"
```

- **Sample Output:**

```
10.625
```

[20]

Problem #AP0203: Lexicographic String Comparison

- **Function:** `int compare_strings(const char* str1, const char* str2)`
- **Description:** Implement a function `compare_strings()` that takes two strings as input and compares them in lexicographic order. The function should return 1 if the first string is larger, 0 if they are equal, and -1 if the first string is smaller.
- **Input:** Two strings `str1` and `str2`.
- **Output:** An integer: 1, 0, or -1.
- **Sample Input:**

```
"apple", "banana"
```

- **Sample Output:**

```
-1
```

[25]

Problem #AP0204: Find Common Integers in Files

- **Function:** `int* find_commons(const char* file1, const char* file2, int* common_count)`
- **Description:** Given two files `file1.txt` and `file2.txt`, each containing non-repeating *unsorted* integer values, implement a function `find_commons()` that takes the filenames as input and returns an array of *sorted* integers that are common in both files. The common elements should also be displayed on the terminal and saved in a new file `file_common.txt`.
- **Input:** Filenames `file1` and `file2`.
- **Output:** An array of integers containing the common elements. The count of common elements should be stored in `common_count`.
- **Sample Input Files:**
  `file1.txt`

```
1 5 2 3 4
```

  `file2.txt`

```
3 4 5 6 7
```

  Sample Output:

```
3 4 5
```

  The output should also be saved in a file `file_common.txt`.

[50]

Problem #AP0205: Student Data Sorting by Marks

- **Function:** `void sort_students(const char* input_file, const char* output_file)`
- **Description:** Given a file `input_file.txt` where each line stores a student's name, roll number, and marks, implement a function `sort_students()` that reads the data into an array of structures. Each structure should store a student's name (at most 30 characters), roll number, and marks. The function should then sort the array with respect to marks and write the sorted data to a new file `output_file.txt`. The number of students is not known beforehand.

- **Input:** Filenames `input_file` and `output_file`.
- **Output:** A new file `output_file.txt` containing the sorted student data.
- **Sample Input File:** `input_file.txt`

```
Subhadeep 101 75.3
Sariful 102 95.8
Barnima 103 85.5
```

- **Sample Output File:** `output_file.txt`

```
Sariful 102 95.8
Barnima 103 85.5
Subhadeep 101 75.3
```

[40]

Problem #AP0206: Substring Search and Replace

- **Function:** `char* find_and_replace(const char* long_string, const char* target_string, const char* new_string)`
- **Description:** Implement a function `find_and_replace()` that takes a long string as input from a file `input_string_long.txt`, along with two other strings `target_string` and `new_string`. The function should replace all occurrences of `target_string` within `long_string` with `new_string`. The function returns the modified string to the main function, which then stores the modified string in a file `output_string_long.txt`.
- **Input:**
  1. `long_string`: A string read from the file `input_string_long.txt`.
  2. `target_string`: A string to be replaced.
  3. `new_string`: A string that will replace `target_string`.
- **Output:** A new string with all occurrences of `target_string` replaced by `new_string`, which is then saved in the file `output_string_long.txt`.
- **Sample Input Files:**
  - `input_string_long.txt`:

    ```
    This is a sample string. This string is used for testing.
    ```

  - `target_string`:

    ```
    ``string"
    ```

  - `new_string`:

    ```
    ``text"
    ```

- **Sample Output File:** `output_string_long.txt`:

```
This is a sample text. This text is used for testing.
```

[50]