Data and File Structures

Assignments

Note:

1. *All assignments are compulsory and should be named in lowercase as in iit2008456_03.c.*
2. *Assignments will be verified by software, so md5sum correct and the o/p should be exactly as indicated.*
3. *Each assignment carries: -5 for copied as detected by standard plagiarism detection software(s),*
   0 for wrong or for not submitted/md5sum mismatch/wrong filename, and
   +5 for correct non-plagiarized program with correct filename.

Problems

1. Implement Merge-Sort in **C** for *n* numbers.
   Eg:
   Input:
   ```
   3
   5
   2
   3
   ```
   Output:
   ```
   2
   3
   5
   ```

2. Implement evaluation of postfix expression in **C++**.
   Eg:
   Input:
   ```
   100 + ( 6 * 4 – ( 8 / 2 ^ 2 ) * 2 ) * 4
   ```
   Output:
   ```
   180
   ```

3. Implement an array-based queue that enqueues +ve integers, dequeues and prints for –ve integers and stops for 0 in **C**. The size of the array is 5.
   Eg:
   Input:
   ```
   5
   -2
   2
   9
   -7
   -7
   0
   ```
   Output:
   ```
   5
   2
   9
   ```

4. Implement Heap-Sort in **C** for *n* numbers.

Eg:
Input:
```
3
5
2
3
```

Output:
```
2
3
5
```

5. Implement an array-based priority queue that enqueues +ve integers, dequeues and prints for –ve integers and stops for 0 in **C++**. The size of the array is 5 and input is in pairs of priority (>0) and name (string).

   Eg:
   Input:
   ```
   2 Gone
   6 With
   -2
   9 Wind
   -7
   -7
   0
   ```

   Output:
   ```
   With
   Wind
   Gone
   ```

6. Implement Binary Search tree (BST) in **C++**. Construct first a BST with the numbers given in the first line. Subsequently, perform **search** (print "found" or "not_found") for (-1 *n*), **insert** for (-2 *i*), delete for (-3 *i*), print minimum for -4, print maximum for -5 and print sorted numbers for -6 and stop for 0.

   Eg:
   Input:
   ```
   5 4 2 6
   -1 4
   -1 3
   -4
   -5
   -6
   0
   ```

   Output:
   ```
   found
   not_found
   2
   6
   2
   4
   5
   6
   ```

7. Implement trie tree in <u>**C**</u>. Insert for (1, *word*), search for (2, *word*), delete for (3, *word*) and stop for 0.

    Eg:

    Input:
    ```
    1 Gone
    1 Wind
    2 Wind
    2 Width
    3 Wind
    2 Wind
    0
    ```

    Output:
    ```
    found
    not_found
    not_found
    ```

8. Implement disjoint set operations. The first *n* lines give elements of *n* sets. Two integers should merge two disjoint sets (stored sequentially, second set is merged at the end of the first) to which they belong and print the representative of disjoint sent. Single integer should result in search for the number and print the representative (first number) if found and -1 otherwise. 0 should result in termination of the program. The program should be written in <u>**C++**</u> but you cannot use any STL container.

    Eg:

    Input:
    ```
    4
    1 3 4 6
    2 7 8 9
    10 11
    12 15 32
    11 32
    4
    9
    10
    20
    0
    ```

    Output:
    ```
    10
    1
    2
    10
    -1
    ```

9. Implement in <u>**C++**</u> Floyd-Warshall algorithm for a given graph in the form of weighted adjacency. Print row-wise the shortest distance between all pairs. Also print path of the shortest distance.

    Eg:

    Input:
    ```
    0 1 5
    1 0 2
    5 2 0
    1 3
    ```

    Output:
    ```
    0 1 3
    1 0 2
    3 2 0
    1 -> 2 -> 3
    ```

10. Implement in **C++** Bredth First Search and print the path length (no. of edges) between nodes *i* and *j*. The graph is undirected and unweighted and the adjacency matrix is given as input.
    Eg:
    Input:
    ```
    0 1 0
    1 0 1
    0 1 0
    1 3
    ```
    Output:
    ```
    2
    ```

11. Implement Depth First Search and print if nodes *i* and *j* are connected ("connected" or "not_connected") until 0 is encountered. The graph is directed and unweighted and the adjacency matrix is given as input. The program should be written in **C++**.
    Eg:
    Input:
    ```
    0 1 0
    1 0 0
    0 1 0
    1 3
    3 1
    0
    ```
    Output:
    ```
    not_connected
    connected
    ```

12. Solve n-queen problem using **C** by printed the number of possible configurations on the chess-board for a given *n*.
    Eg:
    Input:
    ```
    8
    ```
    Output:
    ```
    92
    ```

13. Using hashing by chaining made by an array of 100 linked lists and a hash function that adds the ASCII values and rounds it off to the range 0-99, implement **insert** (1, *word*), **find** (2, *word*) and **delete** (3, *word*). The program should terminate with input 0 and should be written in **C**.
    Eg:
    Input:
    ```
    1 Gone
    1 Wind
    2 Wind
    2 Width
    3 Wind
    2 Wind
    0
    ```
    Output:
    ```
    found
    not_found
    not_found
    ```

14. Solve Fibonocci series finding n^{th} number using recursion and memorization for a given n in <u>**C**</u>.
    Eg:
    Input:
    ```
    8
    ```
    Output:
    ```
    13
    ```

15. Implement Rabin Karp using the hash function given above. The output should be the index of the letter if the substring is found and -1 otherwise. The program is to be written in <u>**C++**</u>.
    Eg:
    Input:
    ```
    ThisIs My Hello World
    Hello
    ```
    Output:
    ```
    11
    ```