Data Structures and Algorithms Projects

1. ADT MultiMap – implementation on a hash table, collision resolution by separate chaining.
2. ADT MultiMap – implementation on a hash table, collision resolution by coalesced chaining.
3. ADT MultiMap – implementation on a hash table, collision resolution by open addressing.
4. ADT MultiMap – implementation on a singly linked list with dynamic allocation.
5. ADT MultiMap – implementation on a singly lined list on an array.
6. ADT SortedMap – implementation on a binary search tree.
7. ADT SortedMap – implementation on a doubly linked list with dynamic allocation.
8. ADT SortedMap – implementation on a hash table, collision resolution by separate chaining.
9. ADT Map – implementation on a hash table, collision resolution by coalesced chaining.
10. ADT Map – implementation on a hash table, collision resolution by open addressing.
11. ADT SortedMultiMap – implementation on a binary search tree.
12. ADT Set – implementation on a hash table, collision resolution by separate chaining.
13. ADT SortedSet – implementation on a binary search tree.
14. ADT Bag – implementation on a doubly linked list on an array.
15. ADT Set – implementation on a hash table, collision resolution by open addressing.
16. ADT Bag – implementation on a hash table, collision resolution by coalesced chaining.
17. ADT SortedBag – implementation on a binary search tree.
18. **Path in a maze.** Consider a maze made of occupied (X) and empty (*) cells. Let R be a robot in this maze.

    Requirements:

    a) Test if R can get out of the maze.
    b) Determine a path to get out of the maze (if there is one).
    c) Determine a path with minimum length to get out of the maze.

    ADTs to be used: Stack (implemented on a singly linked list on an array) and/or Queue (implemented on a doubly linked list on an array) and/or Priority Queue (implemented on a doubly linked list with dynamic allocation) - implement and use at least two ADTs.

19. **Red-Back Card Game**. Two players each receive n/2 cards, where each card can be red or black. The two players take turns; at every turn the current player puts the card from the upper part of his/her deck on the table. If a player puts a red card on the table, the other player has to take all cards from the table and place them at the bottom of his/her deck. The winner is the player that has all the cards. Simulate the game.
    ADTs to be used: Stack (implementation on a singly linked list with dynamic allocation) and Queue (implementation on a doubly linked list on an array).
20. **Evaluation of an arithmetic expression in infix form** (the expression can contain parentheses). The expression contains addition (+), subtraction (-), multiplication (*), division (/), exponentiation (^) and possibly multiple digit operands. The expression will be transformed into postfix notation and the postfix notation will be evaluated.
    ADTs to be used: Stack (implementation on a singly linked list with dynamic allocation) and Queue (implementation on a singly linked list on an array).
21. ADT Priority Queue – implementation on a binary search tree.
22. ADT Priority Queue – implementation on a binary heap.
23. ADT Priority Queue – implementation on a singly linked list on an array.
24. ADT SortedList – implementation on a doubly linked list on an array.
25. ADT SortedList – implementation on a binary search tree.
26. ADT SortedList – implementation on a hash table, collision resolution by separate chaining.
27. ADT SparseMatrix – representation using <line, column, value> triples (value ≠ 0). Implementation on a binary search tree.
28. ADT SparseMatrix – representation using <line, column, value> triples (value ≠ 0). Implementation on a hash table, collision resolution by separate chaining.
29. ADT Set – implementation on a doubly linked list on an array.
30. ADT SortedMultiMap – implementation on a hash table, collision resolution by separate chaining.
31. ADT SparseMatrix - representation using <line, column, value> triples (value ≠ 0). Implementation on a doubly linked list on an array.
32. **Huffman encoding**. Encode/decode a text using Huffman encoding.
    ADTs to be used: Binary Tree (any representation) and Priority Queue (any representation).
33. ADT Binary Tree (with iterators to iterate in preorder, inorder, postorder and level-order). Any representation.