Data Structures and Algorithms Assignment Set – 2

2.1. Define an ADT for Polynomials.

Write C data structure representation and functions for the operations on the Polynomials in a Header file.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

2.2. Define an ADT for Sparse Matrix.

Write C data structure representation and functions for the operations on the Sparse Matrix in a Header file.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

2.3. Define an ADT for List.

Write C data structure representation and functions for the operations on the List in a Header file with array as the base data structure.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file. Two data structures with and without using sentinels in arrays are to be implemented.

2.4. Define an ADT for Set.

Write C data representation and functions for the operations on the Set in a Header file, with array as the base data structure.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

2.5. Given a large single dimensional array of integers, write functions for sliding window filter with maximum, minimum, median, and average to generate an output array. The window size should be an odd integer like 3, 5 or 7. Explain what you will do with the boundary values. (Use zero-padding in both ends of the arrays as necessary.)

Input Output examples for problem no. 2.5

Input: 4, 5, 1, 13, 3, 25, 27, 18, 10, 3, 4, 9

Window size: 3

Max filter output: 5, 5, 13, 13, 25, 27, 27, 27, 18, 10, 9, 9

Min filter output: 0, 1, 1, 1, 3, 3, 18, 10, 3, 3, 3, 0

Median filter output: 4, 4, 5, 3, 13, 25, 25, 18, 10, 4, 4, 4

2.6. Find whether an array is sorted or not, and the sorting order.

2.7. Write a C program to move the negative elements in an array to the front of array.

Example:

Input : 2 -7 10 12 5 -2 32 -4

Output: -7 -2 -4 2 10 12 5 32

2.8. Implement the following functions of ADT Linked List using singly linked list as a header file:

init_l(cur) – initialize a list

empty_l(head) – boolean function to return true if list pointed to by head is empty

atend_l(cur) – boolean function to return true if cur points to the last node in the list

insert_front(target, head) – insert the node pointed to by target as the first node of the list pointed to by head

insert_after(target, prev) – insert the node pointed to by target after the node pointed to by prev

delete_front(head) – delete the first element of the list pointed to by head

delete_after(prev) – delete the node after the one pointed to by prev

2.9. Read integers from a file and arrange them in a linked list (a) in the order they are read, (b) in reverse order. Show the lists by printing by developing a function Print_list. The functions for (a) is Build_list and for (b) is Build_list_reverse.

2.10. Implement the following functions in a menu-driven C program using the data structure operation of Singly Linked List in the header file developed in problem 2.8:

a) print a list (i) in the same order, (ii) in the reverse order.

b) find the size of a list in number of nodes

c) check whether two lists are equal

d) search for a key in (i) an unordered list, (ii) an ordered list (Return the node if key is found and delete the node from original list)

e) append a list at the end of another list.

f) delete the nth node, last node and first node of a list.

g) check whether a list is ordered

h) merge two sorted lists

i) insert a target node in the beginning, before a specified node and at the end of the list (sorted and unsorted).

j) remove duplicates from a linked list (sorted and unsorted)

k) swap elements of a list pairwise

l) move last element to front of a list

m) delete alternate nodes of a list

n) rotate a list

o) delete a list.

p) reverse a list.

q) sort a list.

2.11. Write all the above operations of Single Linked List for the implementation using array. You need to develop Build_list and Build_list_reverse, as well as Print_list.

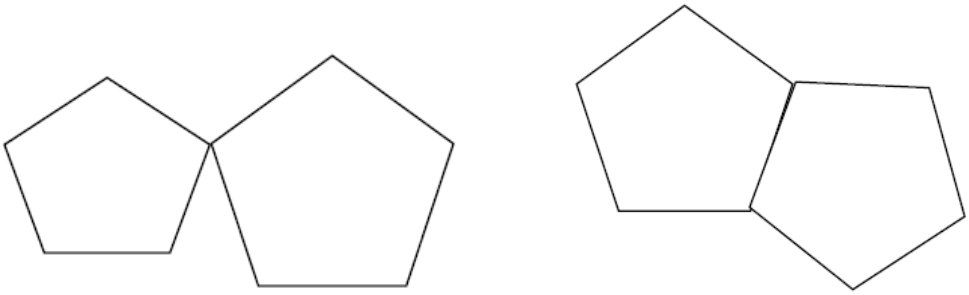
2.12. Repeat problems 2.8 and 2.10 for a circular single linked list, doubly linked list and circular doubly linked list. You need to develop Build_list and Build_list_reverse, as well as Print_list for each case.

2.13. Implement an application to find out the Inverted Index of a set of text files, given a set of keywords. Create a set of 6 text files having the keywords in different positions in the text files. The keywords may occur multiple times in a file. The inverted index file will list the key words along with the file names in which they occur and how many times they occur in that file.

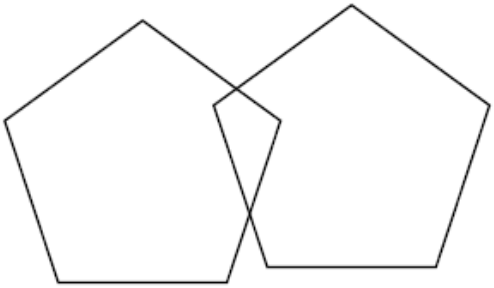
2.14. Write an application for adding, subtracting and multiplying very large numbers, say more than 70-digit integers, using (a) arrays and (b) linked lists to represent the large integers.

2.15. Given two polygons, say pentagons, find out whether they intersect or not.

Touching polygons:



Intersecting polygons:



2.16. Write C language functions for the following sorting algorithms:

Insertion Sort, Selection Sort, Bubble Sort, Mergesort, Quicksort and Heapsort, with all their variants discussed in the class.

Run the functions to sort already-sorted, reverse-sorted and unsorted data from the files of random integers and text strings you have created in Assignment I. Take the first 5000, 10000, 15000, … elements in the array (in sorted, reverse-sorted and unsorted data) and run each of the sorting functions. Find out the number of comparisons made in each case and the execution times taken for each run. Tabulate the data and draw the graphs for discussion on the results obtained vis-à-vis the analysis results.