Lab Assignment No. 7 : Files, Linked Lists, Queues, and Stacks

Exercise 1: Records & Files

Let a record structure defining a patient be given, containing: an identification number, last name, first name, age, gender (male or female), and an illness.
Write a program, using functions and a menu, that allows you to:

1. Create a patient file.
2. Count the number of female patients.
3. Search for a patient by last name.
4. Display the names of patients whose age is greater than a given value.
5. Search for patients whose last names begin with a given letter.
6. Add a patient to the file.
7. Delete a given patient.

Exercise 2: Linked Lists

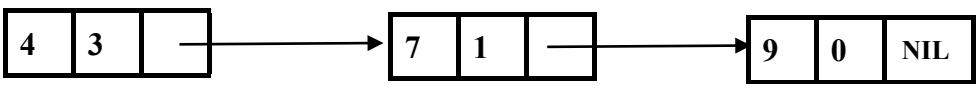
We want to create a circular linked list of integers (the last element points to the first one). Each element contains two fields: value and next. The *value* field contains an integer value and the *next* field contains the address of the next element in the list.

1. Write a function that creates a circular linked list of integers.
2. Write a function that returns 1 if the list is empty, and 0 otherwise.
3. Write a function that counts the number of elements in the list.
4. Write a function that returns the address of the element at the i^{th} position.
5. Write a function that inserts a value V before a value U.
6. Write a function that deletes all elements having a given value.

Exercise 3: Linked Lists, Queues, and Stacks

A polynomial $P(x) = a_n x^n + a_{n-1}x^{(n-1)} + ... + a_1 x + a_0$ can be represented by a simple linked list.

Example: the polynomial $P(x) = 4x^3 - 7x + 9$ can be represented by the following linked list:



1. Define the structure of the linked list.
2. Write a function Insert(L, Coef, Power) which inserts the coefficient *Coeff* and the power *Power* into the list L.
3. Write a function Remove(L, Coef, Power) which removes the first element of the list and retrieves the coefficient *Coeff* and the power *Power*.

4. Write a recursive function Power(x, n).
5. Write a function ReadPoly to read the coefficients of a polynomial of degree n and stores them in a list L.
6. Using the previous Power function, write a function EvaluatePoly that evaluates a polynomial P at a given real point x. This function returns the value of P(x) (the list can be cleared).
7. If we follow the execution of the EvaluatePoly function, we notice that it performs many unnecessary multiplications (when calculating x^{i+1}, it recalculates x^i). One way to avoid this is Horner's method. It consists of noticing that we can write the polynomial in the form:

$$P(x) = (((\dots(a_n x + a_{n-1})x + \cdots)x + a_2)x + a_1)x + a_0$$

Write a function Horner that evaluates the polynomial P.

printf("It is by trying again and again that one finally succeeds. ");