Data Structures and Fundamentals of Programming

Problem #1

In C++, implement a generic **singly-linked-list** class, called `List`, that uses dynamic memory allocation. It should be generic on the type of data stored in the list. This should implement the list ADT. The list should look something like the following:

`beginning -> X0 ->X1 -> … -> Xn-1 <- ending`

where X_0 is the first node in the list and X_{n-1} is the last node in the list. Besides `List`, you will most likely want another generic class called `node`. Along with the class definition(s), you **must** implement the following methods for the generic `List` class:

- Default constructor
- Destructor
- Copy-constructor
- Assignment operator using standard copy semantics
- A method `length` that returns the number of nodes in a list
- Overload the `operator[]` to return the value of the i^{th} element in the list
- `AddToBack` that takes a parameter of item type and creates a new node that is added to the ending of the list
- `RemoveFromFront` that removes a node from the beginning of a list and returns its contents.

Note: Your implementation can **NOT** use STL or any other libraries (standard or otherwise).

Problem #2

In C++, implement a generic **free** function, `removeDuplicates`, that finds all duplicates elements in a given `List` object (from problem 1) and returns a `List` with the duplicates removed. For example, given the `List<int>`: {`1, 2, 3, 4, 2, 3, 3`} the function will return the `List` {`1, 2, 3, 4`}.

The function will be generic on the type of data stored in the `List<T>` object. It will take as parameters a `List<T>` and a comparison operator (i.e., a functor) that takes two parameters of type `T` and returns true if the two elements are equal and false otherwise. The function, `removeDuplicates`, cannot modify any of the actual parameters.

Note: Your implementation can NOT use STL or any other libraries (standard or otherwise).

Problem #3

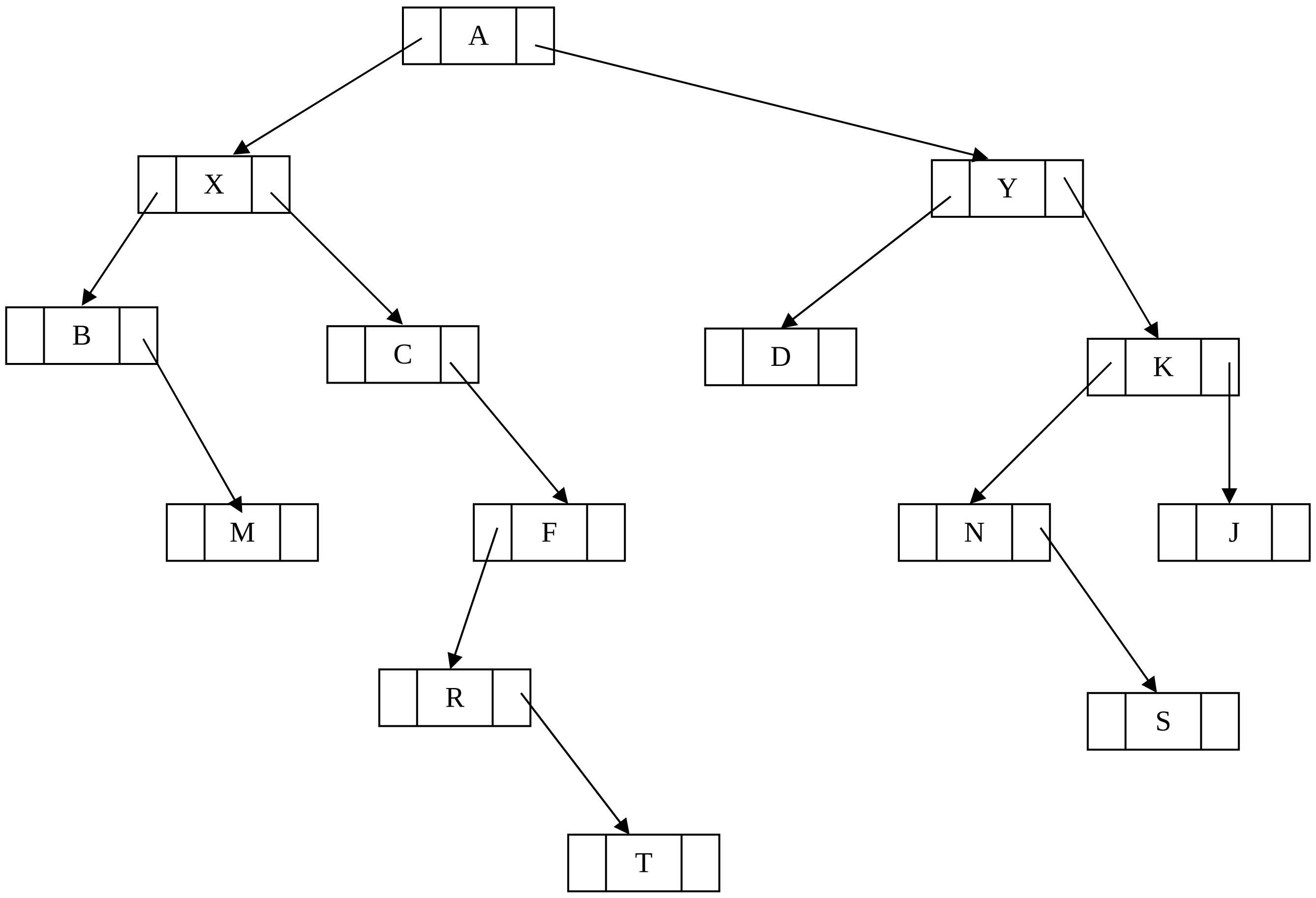
A) Convert the following infix expressions into the equivalent postfix and prefix expressions.

```
a * b - c * d * e * d - f + g

a + b * c * (d - e) - d * f
```

B) Write in, pseudo-code, describe the **preorder**, **postorder**, and **inorder** traversal algorithms. Also give the preorder, postorder, and inorder traversals of the tree below.

C) What is the relationship between the expression notations and the traversal algorithms?


A
X
Y
B
C
D
K
M
F
N
J
R
S
T