Data Structures and Fundamentals of Programming

Problem #1

In C++ implement a *list/iterator* using a generic **double-linked-list** that uses dynamic memory allocation. The list must look like the following:

```
beginning -> X0 <->X2 <-> … <-> Xn <- ending
```

where X_0 is the first node in the list and X_n is the last node in the list. Besides the class called `List`, you will need class called `dnode`. Along with the class definition(s), you must implement the following methods, using standard semantics, for `List<T>`:

- `List()` - Default constructor
- `~List()` - Destructor
- `List (const List<T>&)` - Copy-constructor
- `append (const T&)` – appends an item at the end of the double-linked list
- `remove (const T&)` – removes a node with the item from the double-linked list
- `dnode* <T> search (const T&)`- finds/returns a node with the item from the double-linked list
- `dnode* <T> begin()` – returns the first node of the list
- `dnode* <T> end()` – returns the last node of the list

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

Problem #2

In C++ implement a **ternary tree** abstract data type (ADT) that uses **dynamic memory allocation**. Make it a tree of integers. Each node will have between 0 and 3 children (`left`, `middle`, `right`). Along with the class definition(s), you must implement the following methods for the class `ternary`:

- Default constructor
- Destructor – **must** be recursive or use a recursive method to delete all the nodes in a tree.
- Copy-constructor – **must** be recursive or use a recursive method make a complete copy of a tree.
- `Preorder()` – which prints out the entire tree using a preorder traversal. **Must** be recursive.
- `Postorder()` – which prints out the entire tree using a postorder traversal. **Must** be recursive.

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

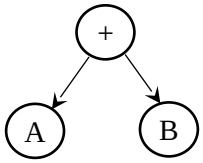
Problem #3

(a) Please first convert each of the following two infix expressions into a binary expression tree and then give their postfix and prefix notations.

B * I + (N + A) / R * Y – E – X * P

P – R * E / (L – I) * (M – I) + N / A / R / Y

(**Hint:** draw an expression tree A+B is like this:



)

(b) Give the preorder, postorder, and inorder traversals of the tree below (use commas to separate different nodes):