Data Structures and Fundamentals of Programming

Problem #1

In C++ implement a **generic** class, called `Stack<T>`, that uses a **single-linked list** implementation. This should implement the stack ADT. It should be generic on the type of the data to be stored. Give all class definitions and implement the following for `Stack`:

- Default constructor
- Destructor
- Copy-constructor
- Assignment operator – using standard copy semantics
- `push(T)` – takes an parameter of type T and adds it to the stack
- `T pop()` – removes a node from the stack

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

Problem #2

In C++ implement a **binary search tree** ADT that uses **dynamic memory allocation**. Make it a simple tree of integers. Along with the class definition(s), you **must** implement the following methods for the class:

- Default constructor
- Destructor
- Copy-constructor
- `insert` which takes a parameter of type integer and creates a new node that is added to the tree in the correct position based on the rules of a binary search tree.

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

Data Structures and Fundamentals of Programming

Problem #3

We define an English word as *reducible* if it is possible to cross out one of its letters and still have an English word and, moreover, it is possible to repeat the process all the way to a single letter. As a simple example, the word "cats" is reducible because you can cross out first the s, then the c, and then the t, leaving the words cat, at, and a, in order. As a more extensive example, the longest reducible word in the EnglishWords.dat lexicon is complecting (the process of joining by weaving or binding together), which survives the following chain of deletions (some of which are admittedly unusual words):

complecting
completing
competing
compting
comping
coping
oping
ping
pig
pi
i

Write a recursive function:

```
bool isReducible(const string&, const Lexicon&);
```

that takes an English word and a lexicon of English words and determines whether the word is reducible.

The class `Lexicon` has a method `bool contains(const string&) const;` that returns `true` if the string represents an English word and `false` otherwise.