```
class Node:
    """
    A structure representing a node inside our binary search tree
    the value is the token,
    the code is the position in the symbol table
    right and left are possible child nodes
    """
    def __init__(self, value, code):
        self.value = value
        self.code = code
        self.right = None
        self.left = None
```

This item is to represent the node inside a binary search tree

```
class SymbolTableBinarySearchTree(SymbolTable):

    def __init__(self):
        self.__root = None
        self.__size = 0

    def pos(self, token):
        """
        Returns the position of the token if the token is present return it's position in the table
        otherwise insert it and return the position
        The table is stored using a binary search tree
        :param token: integer or string the token to be inserted or queried
        :return: int - the position inside the symbol table
        """
        current = self.__root
        parent = None
        """ Search for the token inside the tree"""
        while current is not None and current.value != token:
            parent = current
            if token < current.value:
                current = current.right
            else:
                current = current.left
        """ Handling the cases for the search"""
        if current is None and parent is None: """ If the tree has no nodes"""
            self.__size += 1
            self.__root = Node(token, self.__size) """ The moment when the item is inserted the position  will correspond """
            return self.__size                      """ to the size at the moment so it will stay consistent"""
        elif current is None: """ If the token is not in the tree but the tree has some elements, add it to the tree """
            self.__size += 1    """ and return the  position inside the symbol table """
            if token < parent.value:
                parent.right = Node(token, self.__size)
            else:
                parent.left = Node(token, self.__size)
            return self.__size
        else:
            return current.code """ If it is found return the position """
```