Assignment-3- Problem 1

```
class Queue {
    constructor() {
        this._list = new DLinkedList();
    }

    enqueue(elem) {
        this._list.insertLast(elem);
    }

    dequeue() {
        if (this._list.isEmpty()) {
            throw new Error("Queue is empty");
        }
        return this._list.remove(this._list.first());
    }

    isEmpty() {
        return this._list.isEmpty();
    }

    front() {
        if (this._list.isEmpty()) {
            throw new Error("Queue is empty");
        }
        return this._list.first().element();
    }

    size() {
        return this._list.size();
    }
}
```

Problem 2:

Algorithm Description

Algorithm for method `isBalanced`

1. Create an empty stack of characters.
2. Assume that the expression is balanced (`balanced` is `true`).
3. Set index to 0.
4. `while balanced` is `true` and index < the expression's length
5. Get the next character in the data string.
6. `if` the next character is an opening parenthesis
7. Push it onto the stack.
8. `else if` the next character is a closing parenthesis
9. Pop the top of the stack.
10. `if` stack was empty or its top does not match the closing parenthesis
11. Set balanced to `false`.
12. Increment index.
13. Return `true` if `balanced` is `true` and the stack is empty.

Pseudo Code

```
Algotithm isExpressionValid() {
    int n = expression.length(); // get max size (chars) of expression
    ArrayStack stk = new ArrayStack(n); // create stack
    char ch, chx; // ch: char scanned and chx: char popped
    boolean valid = false;
    for i = 0 to n-1 do // get a char until □n□ chars are scanned
    {
      ch = expression.charAt(i); // get char
      if (ch == '[' || ch == '{' || ch == '('
                stk.push(ch);
      if (ch == ']' || ch == '}' || ch == ')') {
        if (stk.isEmpty())
            valid = false;
        else {
          chx = (Character) stk.pop(); // pop a char from stack
```

```
            if ((chx == '[' && ch == ']') || (chx == '{' && ch == '}') || (chx == '(' && ch == ')') )
               valid = true;
            else
            {
               valid = false;
                return valid;
            }

        }
      }
      }
      if (!stk.isEmpty()) // stack not empty
         valid = false;
      return valid;
   }
}
```