List Data Structure

List is implemented using an array, and several operations are defined to manipulate the list.

Key Characteristics of a List:

1. **Ordered Collection:**
   The elements in a list are stored in a specific order, and this order is important.
   Example: The list (2, 6, 8, 7, 1) has a specific sequence.

2. **Homogeneous Elements:**
   All elements in the list are of the same type (e.g., integers, strings, etc.).

3. **Dynamic Operations:**
   Elements can be inserted, removed, or updated at any position in the list.

List Operations:

1. `createList()`

- Creates a new list, typically initializing it as empty.
- This function sets up the initial state of the list, often by allocating memory for the array and setting the size to zero.

---

2. `add(X)`

- Inserts an element **X** at a specific position in the list.
- If the list is implemented using an array, inserting an element may require shifting elements to the right to make space for the new element.
  Example: Inserting 9 into the list `(2, 6, 8, 7, 1)` at position 3 results in `(2, 6, 8, 9, 7, 1)`.

---

3. `remove()`

- Removes the element at the current position in the list.

- After removal, the elements to the right of the removed element are shifted left to fill the gap.
  Example: Removing the element at position 4 from the list `(2, 6, 8, 9, 7, 1)` results in `(2, 6, 8, 9, 1)`.

---

4. `get()`

- Retrieves the element at the current position in the list.
  Example: If the current position is 3, `get()` would return 8 from the list `(2, 6, 8, 9, 1)`.

---

5. `update(X)`

- Replaces the element at the current position with a new value **X**.
  Example: Updating the element at position 2 with 5 in the list `(2, 6, 8, 9, 1)` results in `(2, 5, 8, 9, 1)`.

---

6. `find(X)`

- Searches the list for an element **X** and returns its position if found.
- If the element is not found, it returns an indication (e.g., `-1`).
  Example: Searching for 8 in the list `(2, 6, 8, 9, 1)` would return position 2.

---

7. `length()`

- Returns the number of elements in the list.
  Example: The length of the list `(2, 6, 8, 9, 1)` is **5**.

---

8. `start()`

- Moves the current position pointer to the beginning of the list.
  Example: After calling `start()`, the current position points to **2** in the list `(2, 6, 8, 9, 1)`.

---

9. `end()`

- Moves the current position pointer to the end of the list.
  Example: After calling `end()`, the current position points to **1** in the list `(2, 6, 8, 9, 1)`.

---

10. `next()`

- Moves the current position pointer forward by one element.
  Example: If the current position is 2 (pointing to 6), calling `next()` moves it to position 3 (pointing to 8).

---

11. `back()`

- Moves the current position pointer backward by one element.
  Example: If the current position is 3 (pointing to 8), calling `back()` moves it to position 2 (pointing to 6).

---

12. `clear()`

- Removes all elements from the list, effectively resetting it to an empty state.
  Example: After calling `clear()` on the list `(2, 6, 8, 9, 1)`, the list becomes **empty**.
