Lists worksheet

Arjun Chandrasekhar

Review of list operations

The list ADT is a data type that allows the program to store elements in an ordered sequence. We will work with the following fundamental list operations:

- **append(value)**: Adds value to the end of the list.
- **prepend(value)**: Adds value to the beginning of the list.
- **insertAt(position, value)**: Inserts value at the specified position.
- **get(position):** Returns the element at position
- **remove(position)**: Removes and returns the element at position.
- **search(value)**: Returns the position of value in the list or -1 if not found.
- **isEmpty()**: Returns true if the list contains no elements, false otherwise.
- **getLength()**: Returns the number of elements in the list.

Tracking list operations

Given an initially empty List ADT, show the resulting state of the list after the following sequence of operations:

- append(3)
- append(7)
- prepend(1)
- insertAt(2, 5)
- prepend(10)
- prepend(11)
- insertAt(4, 10)
- remove(5)

Manipulating lists

Give a series of list operations to transform the list [2, 4, 6, 8] into [19, 1, 2, 2, 13, 4, 8, 6, 7, 7, 8]. Use as few list operations as possible.

Using List to implement Stack

Recall the following operations of the Stack ADT:

- **Push:** add a new item to the top of the stack
- **Pop:** remove and return the item at the top of the stack
- **Peek:** return the value of the item at the top of the stack (leaving it there)

Let's explore how we might implement the Stack ADT using the (more flexible) List ADT. Below is the outline of a Stack class that uses a List object to represent the stack (rather than a static array or a chain of links). Fill in the push, pop, and peek methods. **These methods should be very easy to implement, given that List is a more flexible version of Stack!**

```
class Stack
{
    List stack; // use a List object to store the stack elements

    push(data)
    {
        // fill in how we should implement push using List operations




    }

    pop()
    {
        // fill in how we should implement pop using List operations




    }

    peek()
    {
        // fill in how we should implement peek using List operations




    }
}
```

Using List to implement Queue

Recall the following operations of the Stack ADT:

- **Enqueue:** add a new item to the back of the queue
- **Dequeue:** remove and return the item at the front of the queue
- **Peek:** return the value of the item at the front of the queue (leaving it there)

Let's explore how we might implement the Queue ADT using the (more flexible) List ADT. Below is the outline of a Queue class that uses a List object to represent the stack (rather than a static array or a chain of links). Fill in the push, pop, and peek methods. **These methods should be very easy to implement, given that List is a more flexible version of Stack!**

```
class Queue
{
    List queue; // use a List object to store the queue elements

    enqueue(data)
    {
        // fill in how we should implement enqueue using List operations




    }

    dequeue()
    {
        // fill in how we should implement dequeue using List operations




    }

    peek()
    {
        // fill in how we should implement peek using List operations




    }
}
```