

Director Lists Notes

I. Introduction

A. Reference: **Director 7 and Lingo Bible**, chap 12.

B. Two types of lists:

1. Linear list
2. Property list

C. Each can be of two types:

1. Sorted
2. Unsorted

D. Once a list has been sorted with the **sort** command, the list remains sorted even when you make changes to it.

E. Operations:

1. Add data at beginning of or at a specific location in a list
2. Append data at the end of a list
3. Remove data from a list.
4. Access data from a specific location in a list
5. Edit data stored in a list
6. Sort data in a list
7. Count the number of data items in a list

F. General rules:

1. In all cases the **myList** argument refers to the list you are modifying.
2. If the data you add is a string, must enclose within double quotation marks. If data is numeric, do **not** use quotation marks.

II. List-related Lingo

A. See **table 12-1**.

B. Syntax:

1. add	linear lists
2. addAt	linear
3. addProp	Property lists
4. append	Linear
5. count	both
6. deleteAt	both
7. deleteOne	both
8. deleteProp	Property
9. duplicate()	both
10. findPos()	both
11. findPosNear()	both
12. getAProp()	Property
13. getAt()	both
14. getLast()	both
15. getOne()	both
16. getPos()	both
17. getProp()	Property
18. getPropAt()	Property
19. ilk()	both
20. list()	linear
21. listP()	both
22. setAProp	Property
23. setAt	both
24. setProp	Property
25. sort	both
26. max	both
27. min	both
28. []	both
29. . (dot)	both

III. Working with Linear Lists

A. A list must have a name and must be initialized with or without data.

1. You declare or initialize a list when you use it.
2. Two forms:

a) set myList to [item1, item2, ... itemN]

b) set myList to list(item1, item2, ..., itemN) This is rarely used.

B. Example:

1. set products to ["motherboard", "chip set", "keyboard"]
2. set products = ["motherboard", "chip set", "keyboard"]
3. products = ["motherboard", "chip set", "keyboard"]

C. Can do same for list function:

1. set products to list("motherboard", "chip set", "keyboard")
2. set products = list("motherboard", "chip set", "keyboard")
3. put list("motherboard", "chip set", "keyboard") into products
4. products = list("motherboard", "chip set", "keyboard")

D. Can also put numbers in a list:

1. playerScores = [50, 27, 33, 66]

E. Lists are **not** homogeneous:

1. set person to ["John Smith", 34, #sales]
2. Property lists usually used to hold values of different types.

F. To initialize an empty list:

1. set products to []
2. put [] into products
3. products = []

IV. Getting values

A. All of the list functions assume that the first element in a list is element number **1** (not 0).

B. getAt() and []

1. Syntax: **myList** is the name of the variable, **index** holds a number.

- a) getAt(myList, index)
- b) myList.getAt(index)
- c) myList[index]

2. Example:

- a) daysOfWeek = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat",]
- b) put getAt(daysOfWeek, 3)
- c) -- "Tue"

- d) put daysOfWeek.getAt(4)
- e) -- "Wed"

- f) put daysOfWeek[5]
- g) -- "Thu"

- h) thirdDay = getAt(daysOfWeek, 3)
- i) put thirdDay
- j) -- "Tue"

- k) eightDay = daysOfWeek[8]
- l) -- ERROR!! Director **does** check bounds!

3. Lists of lists:

- a) myTable = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
- b) To get the 23:
- c) put getAt (getAt (myTable, 2), 3)
- d) or:
- e) temp = getAt(myTable, 2)
- f) getAt(temp, 3)
- g) or
- h) put myTable[2] [3]

C. getLast()

1. Syntax:

- a) getLast (mylist)
- b) myList.getLast()

2. Examples:

- a) put getLast(daysOfWeek)
- b) -- "Sat"
- c) put daysOfWeek.getLast ()
- d) -- "Sat"

D. getOne()

1. Returns the position of an item in a list.

2. Syntax: **aValue** is the value that you expect to be in the list. Returns the index of **aValue** in **myList**.

- a) getOne(myList, aValue)
- b) myList.getOne(aValue)

3. Examples:

- a) put getOne(daysOfWeek, "Tue")
- b) -- 3
- c) put getOne(daysOfWeek, "Tuesday")
- d) -- 0
- e) put daysOfWeek.getOne("Sat")
- f) -- 7

E. getPos()

1. Same as **getOne()**

V. Setting Values

A. Means changing a value at a particular location to a new value.

B. SetAt

1. Syntax:

- a) `setAt myList, index, aValue`
- b) `myList.setAt(index, aValue)`
- c) `myList[index] = aValue`

2. Examples:

- a) `pets = ["dog", "cat", "fish"]`
- b) `setAt pets, 3, "bird"`
- c) `put pets`
- d) `-- ["dog", "cat", "bird"]`
- e) `pets.setAt(3, "hamster")`
- f) `put pets`
- g) `-- ["dog", "cat", "hamster"]`

3. Note: no parenthesis used with `setAt`. This is because it is a command not a function. Parenthesis are optional.

4. Must use parenthesis when use dot notation.

5. Example:

- a) `setAt(pets, 3, "newt")`
- b) `put pets`
- c) `-- ["dog", "cat", "newt"]`
- d) `setAt(pets, 7, "snake")`
- e) `put pets`
- f) `-- ["dog", "cat", "newt", 0, 0, 0, "snake"]`

6. Setting a position outside range changes size of the list.

7. Using []:

- a) `pets = ["dog", "cat", "hamster"]`
- b) `pets[1] = "iguana"`
- c) `put pets`

d) -- ["iguana", "cat", "hamster"]

VI. Adding to a List

A. `addAt` **inserts** a value at a given position. Shifts all other items down one.

B. Syntax:

1. `addAt , myList, index, aValue`
2. `myList.addAt(index, aValue)`

C. Example (from above):

1. `pets = ["fish", "dog", "cat", "hamster"]`
2. `addAt pets, 3, "bird"`
3. `put pets`
4. -- ["fish", "dog", "bird", "cat", "hamster"]

5. `pets.AddAt (1, "newt")`
6. `put pets`
7. -- ["newt", "fish", "dog", "bird", "cat", "hamster"]

D. Lingo pads if use index outside range:

1. `pets = ["fish", "dog"]`
2. `addAt(pets, 5, "newt")`
3. `put pets`
4. -- ["fish", "dog", 0, 0, "newt"]

E. If a list has been sorted, `addAt` breaks the sorting

1. `pets = ["cat", "dog"]`
2. `sort pets`
3. `put pets`
4. -- ["cat", "dog"]
5. `pets.addAt(2, "bird")`
6. `put pets`
7. -- ["cat", "bird", "dog"]

F. `add` Places the item into the list at the end. Unless the list is sorted, then places in

correct position.

1. Syntax:

- a) add myList, aValue
- b) myList.add(aValue)

2. Example:

- a) pets = []
- b) add pets, "dog"
- c) pets.add("cat")
- d) put pets
- e) -- ["dog", "cat"]

3. Example: get a quick list of the names of all the members in a castLib:

```
on GetMemNames whichCast
  nameList = [ ]
  repeat with i = 1 to the number of members of castLib whichCast
    add nameList, the name of member i of castLib whichCast
  -- Alternate syntax for the above:
  -- add nameList, member(i, whichCast).name
  end repeat
  return nameList
end
```

4. Example with sorted list:

- a) sort pets
- b) put pets
- c) -- ["cat", "dog"]
- d) pets.add("bird")
- e) put pets
- f) -- ["bird", "cat", "dog"]

G. append. always puts the value at the end of the list, even if it is sorted.

VII. Deleting items from linear lists.

A. deleteAt. opposite of addAt. Deletes the item at a specific position.

1. Syntax:

- a) deleteAt myList, index
- b) myList.deleteAt(index)

2. Example:

- a) pets = ["cat", "dog", "fish", "hamster"]
- b) deleteAt pets, 2
- c) put pets
- d) -- ["cat", "fish", "hamster"]

B. deleteOne. Remove item from list based on its value.

1. Syntax:

- a) deleteOne myList, aValue
- b) myList.deleteOne(aValue)

2. Example:

- a) pets = ["cat", "dog", "fish", "hamster"]
- b) deleteOne pets, "cat"
- c) put pets
- d) -- ["dog", "fish", "hamster"]

3. If there are two items with same name, deletes the first. To remove all instances, must write a script:

```
on Removeall myList, anItem
  repeat while getOne(myList, anItem)
    deleteOne(myList, anItem)
  end repeat
end
```

4. get the list back from the handler automatically since lists are pointers. See below.

VIII. Transferring and duplicating lists

A. Variables are kept as **values**.

1. intA = 12
2. intB = intA
3. put intB
4. -- 12
5. intB = 7
6. put intB
7. -- 7
8. put intA
9. -- 12

B. Lists are kept as **pointers**.

1. listA = [1, 2, 3, 4, 5]
2. listB = listA
3. add listB, 6
4. put listB
5. -- [1, 2, 3, 4, 5, 6]
6. put listA
7. -- [1, 2, 3, 4, 5, 6]

C. Both listB **and** listA change! If want a copy, must duplicate the list:

1. listA = [1, 2, 3, 4, 5]
2. listB = duplicate(listA)
3. add listB, 6
4. put listB
5. -- [1, 2, 3, 4, 5, 6]
6. put listA
7. -- [1, 2, 3, 4, 5]

D. Can use dot notation: listB = listA.duplicate()

E. Problem: copying a list can be very **slow**.

F. Parameters are **pass by value**. But passing a list is passing a **pointer** so when you change a list in a handler, you change the original list!

G. You **can** return the list anyway. Makes the removeAll code easier to use:

```
on Removeall myList, anItem
  repeat while getOne(myList, anItem)
    deleteOne(myList, anItem)
  end repeat

  return myList
end
```

H. To use this:

1. pets = ["cat", "dog", "cat"]
2. pets = RemoveAll(pets, "cat")
3. put pets
4. -- ["dog"]

IX. Property Lists.