ChunkList

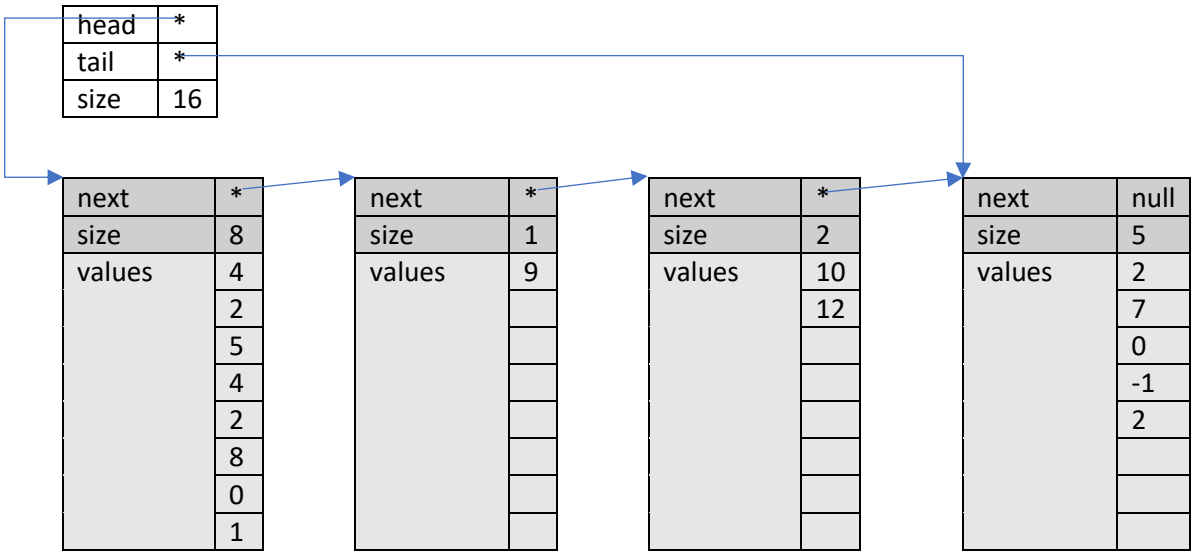
Original ChunkList concept by Nick Parlante. Adapted to an assignment by Varick Erickson.

1 INTRODUCTION

A ChunkList is like a regular linked list, except each node contains a little fixed size array of elements instead of just a single element. Each node also contains its own "size" int to know how full it is.

The ChunkList will have the following features...

- The ChunkList object contains a head pointer to the first chunk, a tail pointer to the last chunk, and an int to track the logical size of the whole collection. When the size of the list is 0, the head and tail pointers are null.



- Each chunk contains a fixed size ItemType[] array, an int to track how full the chunk is, and a next pointer. There should be a constant ARRAY_SIZE = 8 that defines the fixed size of the array of each chunk. Elements should be added to the array starting at its 0 index. Elements in each little array should be kept in a contiguous block starting at 0 (this will require shifting elements around in the little array at times). You may want to test ARRAY_SIZE set to smaller values, but turn it in with ARRAY_SIZE set to 8.

- The empty collection should be implemented as null head and tail pointers. Only allocate chunks when actually needed.

- ChunkList should implement the following methods similar to those describe on pages 136-138:
    - MakeEmpty
    - isFull → (refers to the entire list, not an individual chunk node)
    - GetLength → (refers to the total number of element, not the number of chunk nodes)
    - GetItem
    - PutItem
    - DeleteItem
    - ResetList
    - GetNextItem

- The PutItem operation should add new elements at the end of the overall collection - i.e. new elements should always go into the tail chunk. If the tail chunk is full, a new chunk should be added and become the new tail. We are not going to trouble ourselves shifting things around to use empty space in chunks in the middle of the list. We'll only look at the tail chunk.

- The DeleteItem operation should look through each chunk array until the target element is found. Since the ChunkList can potentially have duplicates of the same elements, DeleteItem should just delete the first found instance of the element. If the chunk node is empty after removing the element, then the entire chunk should be removed from the link list.

- Do not use a dummy node because (a) it does not help the code much, and (b) dummy nodes are lame.

- Keep a single "size" variable for the whole list that stores the total number of client data elements stored in the list. Similarly, keep a separate size in each chunk to know how full it is.

2 TEST DRIVER

You should have a test driver similar to the list driver demonstrated in class. Much of this driver can be reused for testing you implementation of ChunkList. In particular, it should support the following commands:

- GetLength
- PutItem
- DeleteItem
- IsFull
- MakeEmpty
- PrintList

IMPORTANT: Be sure to craft your input tests to ensure that your node removal works correctly.

3 QUESTIONS

1. What is the advantage of the ChunkList approach as opposed to the Unsorted link list implementation given in Chapter 3?
2. What would be the implications of increasing the size of ARRAY_SIZE to a very large value?
3. What are the running times of:
    - MakeEmpty
    - isFull
    - GetLength
    - GetItem
    - PutItem
    - DeleteItem
    - ResetList
    - GetNextItem

4. Compare placing a new element into the FIRST available empty space versus placing a new element in the tail chunk. What are the advantages and disadvantages to automatically placing values at the tail node?
5. How could you create a linked list entirely on the stack?

4 DELIVERABLES

- Your ChunkList class
- Pre and post conditions for the methods
- Your test driver
- Your test plan as input to the test driver
- The output file from the test driver
- Answers to the questions for part 3