Linked Lists

Lecture 16
Sections 18.1 - 18.3

Robb T. Koether

Hampden-Sydney College

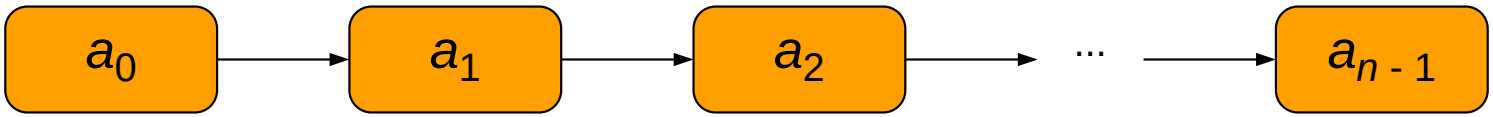
Mon, Feb 19, 2018

Outline

Linked Lists

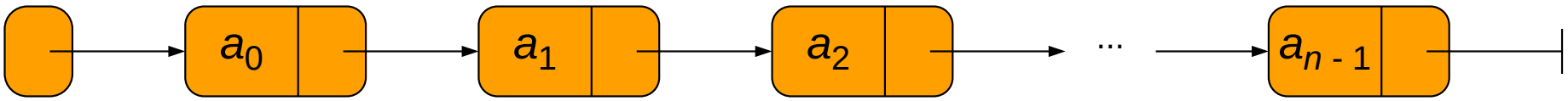
- In a linked list, each node is allocated dynamically when the element is added to the list and deallocated dynamically when it is removed from the list.
- A linked list always uses exactly the amount of memory it needs.
- A linked list is more efficient than an array list in some ways and less efficient in others.
- The name of the linked list class is `LinkedList`.

Linked Lists



The abstract concept

Linked Lists



The linked implementation

Outline

The `LinkedListNode` Class

- We create a separate class, `LinkedListNode`, as a data type that stores a single element of a linked list.
- Each `LinkedListNode` contains a data item and a pointer that links it to the next node.

`LinkedListNode` Data Members

`LinkedListNode` Data Members

- `T m_value;`
- `LinkedListNode* m_next;`

- `m_value` – The element a_i.
- `m_next` – A pointer to the node containing the element a_{i+1} or `NULL` if a_i is the last element.

`LinkedListNode` Member Functions

`LinkedListNode` Public Member Functions

- `T value() const;`
- `T& value();`
- `LinkedListNode* next();`

- `T value()` – Returns a copy of the data member of the node (*r*-value).
- `T& value();` – Returns a reference to the data member of the node (*l*-value).
- `next();` – Returns a *copy* of the pointer to the next node.

`LinkedListNode` Member Functions

`LinkedListNode` Private Member Functions

- `LinkedListNode(const T& val = T());`

- `LinkedListNode(T&);` – Constructs a linked list node containing the specified value.
- It can be invoked only by friend classes.
- The only friend class (for now) will be the `LinkedList` class.

`LinkedListNode` Destructor

`LinkedListNode` Destructor

- `~LinkedListNode();`

- What should the `LinkedListNode` destructor do?

The `LinkedListNode` Class

The `LinkedListNode` Class

- The header file `linkedlistnode.h`.

Outline

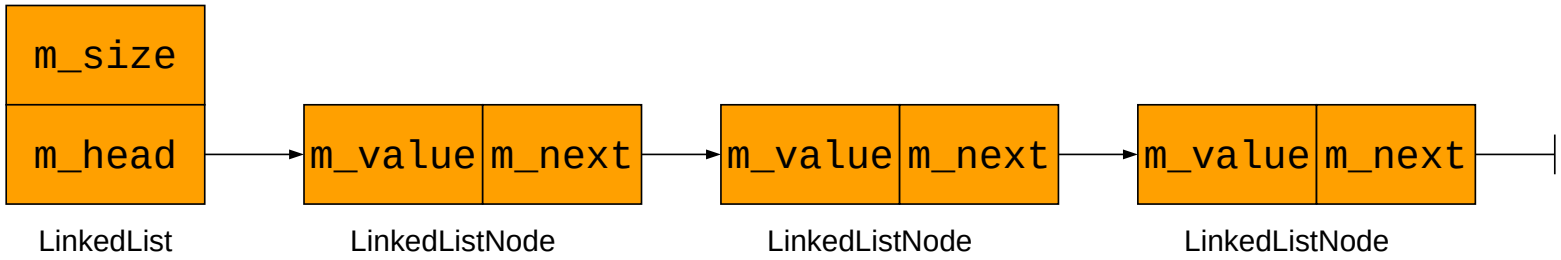
The `LinkedList` Class

`LinkedList` Data Members

- `int m_size;`
- `LinkedListNode* m_head;`

- `m_size` – The number of elements in the list.
- `m_head` – A pointer to the first node (which contains the first element).

The `LinkedListNode` Class



Validity Requirements of a Linked List

- The linked list structure is valid provided
  - `m_size >= 0`.
  - If `m_size == 0`, then `m_head == NULL`.
  - If `m_size > 0`, then `m_head != NULL`.
  - For every `i` from `0` to `m_size - 2`, in node `i`, `m_next != NULL`.
  - In node `m_size - 1`, `m_next == NULL`.

Outline

Random Access vs. Sequential Access

- A linked list allows sequential access, but not random access.
- To find the list element in position n, we must begin at the head of the list and move sequentially through positions $1, 2, \ldots, n-1$ before reaching position n.
- An array allows random, or direct, access.

Random Access vs. Sequential Access

- Sequential access is much slower than random access if the list elements are accessed randomly.
- However, sequential access may be faster than random access if the list elements are accessed sequentially.

The Technique of Chasing Pointers

Chasing Pointers

```
LinkedListNode* node = m_head;
for (int i = 0; i < pos; i++)
    node = node->m_next;
```

- Technique of chasing pointers to locate position `pos`.

Random Access vs. Sequential Access

Random Access vs. Sequential Access

```
for (int i = 0; i < list.size(); i++)
    list[i] = 0;
```

- How would the performance of the above **`for`** loop differ between the `ArrayList` and `LinkedList` implementations?

The `LinkedList` Class

The `LinkedList` Class

- `linkedlistnode.h`.
- `linkedlist.h`.
- `ListTest.cpp`.

Outline

Assignment

Assignment

- Read Sections 18.1 - 18.3.