**LINKED LIST**

- Linked list contains 0 or more nodes that are linked to one another:

```
list head
```

- Every node stores **data** and a **pointer** to the next node (*ListNode * next)
- List has a **ListNode * head** to point to the **first** node
- Last node must point to **NULL** (address 0)
- How would you traverse a list?
**Traversing a Linked List**

```
void NumberList::printList() const
{
    ListNode * nodePtr = head;
    while (nodePtr) { // while nodePtr != 0
        cout << nodePtr->value << "\t";
        nodePtr = nodePtr->next;
    }
    cout << endl;
}
```

- Every node stores data and a pointer to the next node (`ListNode * next`).
- List has a list head to point to the first node.
- The last node must point to null (address 0).
Traversing a Linked List

nodePtr = head

nodePtr points to the node containing 5, then the node containing 13, then the node containing 19, then points to the null pointer, and the list traversal stops.
TRACINGING A LINKED LIST

nodePtr = nodePtr->next

nodePtr points to the node containing 5, then the node containing 13, then the node containing 19, then points to the null pointer, and the list traversal stops.
Traversing a Linked List

nodePtr = nodePtr->next

nodePtr points to the node containing 5, then the node containing 13, then the node containing 19, then points to the null pointer, and the list traversal stops.
Traversing a Linked List

nodePtr = nodePtr->next

nodePtr points to the node containing 5, then the node containing 13, then the node containing 19, then points to the null pointer, and the list traversal stops.
How would you append an element at the end of the linked list?

Suppose you want to store data in ascending order

- 1 3 4 5 8
- Insert 6:
  - 1 3 4 5 6 8

How would you insert an element into a linked list?
INSERTING A NODE INTO A LINKED LIST

- Used to maintain a linked list in order
- Requires **two pointers** to **traverse** the list:
  - pointer to locate the node with data value greater than that of node to be inserted
  - pointer to 'trail behind' one node, to point to node before point of insertion
- New node is inserted between the nodes pointed at by these pointers
**Inserting a Node into a Linked List**

We need an algorithm to find the appropriate location for the new node...

New node created, correct position located
**Inserting a Node into a Linked List**

---

...And to correctly update all the corresponding pointers

```
*previousNode    nodePtr

list head
```

```
5 13 19

newNode
```

newNode’s next points to next node in list

```
5 13 17 19
```

New node inserted in order in the linked list
void NumberList::insertNode(double num)
{
    ListNode *newNode; // A new node
    ListNode *nodePtr; // To traverse the list
    ListNode *previousNode = nullptr; // The previous node

    // Allocate a new node and store num there.
    newNode = new ListNode;
    newNode->value = num;

    // If there are no nodes in the list
    // make newNode the first node
    if (!head)
    {
        head = newNode;
        newNode->next = nullptr;
    }
    else // Otherwise, insert newNode
    {
        // Position nodePtr at the head of list.
        nodePtr = head;
// Initialize previousNode to nullptr.
previousNode = nullptr;

// Skip all nodes whose value is less than num.
while (nodePtr != nullptr && nodePtr->value < num)
{
    previousNode = nodePtr;
    nodePtr = nodePtr->next;
}

// If the new node is to be the 1st in the list,
// insert it before all other nodes.
if (previousNode == nullptr)
{
    head = newNode;
    newNode->next = nodePtr;
}
else // Otherwise insert after the previous node.
{
    previousNode->next = newNode;
    newNode->next = nodePtr;
}

Program 17-3

```cpp
1 // This program demonstrates the insertNode member function.
2 #include <iostream>
3 #include "NumberList.h"
4 using namespace std;
5
6 int main()
7 {
8     // Define a NumberList object.
9     NumberList list;
10
11     // Build the list with some values.
12     list.appendNode(2.5);
13     list.appendNode(7.9);
14     list.appendNode(12.6);
15
16     // Insert a node in the middle of the list.
17     list.insertNode(10.5);
18
19     // Display the list
20     list.displayList();
21     return 0;
22 }
```

Program Output

2.5
7.9
10.5
12.6
**Inserting a Node**

**Initialization:**
- New node created;
- nodePtr points to 1st element;
- previousNode points to null.

```
previousNode  nodePtr = head

null

list
head

5 -> 13 -> 19 -> null

newNode

17 -> null
```
**Inserting a Node**

```
previousNode  nodePtr = head

null

list head

5 next 13 next 19 next null

1st iteration:
Is nodePtr == null?
Is nodePtr->value > 17?
No. Therefore, update nodePtr and previousNode

newNode

17 null
```
**INSERTING A NODE**

previousNode = nodePtr  

nodePtr = nodePtr->next

1st iteration: 
Update nodePtr and previousNode:

previousNode = nodePtr;
nodePtr = nodePtr->next;

list head

5 next 13 next 19 next null

newNode

17 next null
**INSERTING A NODE**

2nd iteration:
Is nodePtr == null?
Is nodePtr->value > 17?
No. Therefore, update nodePtr and previousNode

previousNode

```
5
next
```

nodePtr

```
13
next
```

```
19
next
null
```

list head

newNode

```
17
null
```

© Pearson Education Limited 2015
INSERTING A NODE

previousNode = nodePtr

nodePtr = nodePtr->next

2nd iteration:
Update nodePtr and previousNode:
previousNode = nodePtr;
nodePtr = nodePtr->next;

list head

5

next

13

next

19

next null

newNode

17

null
3rd iteration:  
Is `nodePtr` == `null`?  
Is `nodePtr`->value > 17?  
Yes (2nd question)  
Therefore, insert the new node
**Inserting a Node**

Insert new node:
What pointers must change?
1) previousNode->next
2) newNode->next

Previous node: 5
Current node: 13
New node: 17
Next node: 19

previousNode

nodePtr

list
head

newNode

null
Inserting a Node

previousNode

nodePtr

5

13

19

list

head

Update
previousNode->next

next

previousNode->next = newNode;
**INSERTING A NODE**

previousNode

nodePtr

list

head

newNode

Update

newNode->next

newNode->next = nodePtr;
What if the new node is smaller than all other nodes?

In this case, new node becomes the new head.
**Inserting a Node into a Linked List**

previousNode

null

nodePtr

newNode

list

head

5

13

19

null

New node created, correct position located
Inserting a Node into a Linked List

Previously, we have:

- previousNode: null
- nodePtr

Now, let's insert a new node:

- newNode
- list head

New node created, correct position located

```
head = newNode;
newNode->next = nodePtr;
```
**INSERTING A NODE INTO A LINKED LIST**

- **Must we deallocate (delete) `previousNode`, `nodePtr`, `newNode` when we’re done inserting a node?**

  - **No:**
    - These variables were used to temporarily store addresses.
    - Now the addresses are permanently stored in the linked list structure.
    - The `tmp` pointer variables can just go out of scope when we exit the function.

![Linked List Diagram](image-url)
Can we insert a node using only one iterator pointer: `nodePtr`?

Yes:
- `nodePtr` will be treated as “previous node”
- `nodePtr->next` will be treated as “next node”
- See example code
INSERTING A NODE

nodePtr

list head

Update:
newNode->next
nodePtr->next

newNode

null

17
null

5
next
13
next
19
next
null

© Pearson Education Limited 2015
INSERTING A NODE

Inserting a node into a linked list.

```c
newNode->next = nodePtr->next;
```

Diagram:
- List head: 5 -> 13 -> 19 (null)
- newNode: 17
- Update `newNode->next`
Inserting a Node

nodePtr

list head

5

13

19

next

next

next

nodePtr->next = newNode;

Update
nodePtr->next

newNode

© Pearson Education Limited 2015