Chapter 17: Linked Lists
17.1

Introduction to the Linked List

ADT
Introduction to the Linked List ADT

- **Linked list**: set of data structures (nodes) that contain references to other data structures

![Diagram of a linked list with nodes connected by arrows, starting from a node labeled 'list head' and ending at a node labeled 'NULL.']
Introduction to the Linked List ADT

- References may be addresses or array indices
- Data structures can be added to or removed from the linked list during execution

![Linked List Diagram]

```c
list head
```

```c
newNode
```
Linked Lists vs. Arrays and Vectors

- Linked lists can grow and shrink as needed, unlike arrays, which have a fixed size.
- Linked lists can insert a node between other nodes easily.
Node Organization

• A node contains:
  – data: one or more data fields – may be organized as structure, object, etc.
  – a pointer that can point to another node
Linked List Organization

• Linked list contains 0 or more nodes:

• Has a list head to point to first node

• Last node points to **NULL**
Empty List

- If a list currently contains 0 nodes, it is the **empty list**
- In this case the list head points to **NULL**
Declaring a Node

- Declare a node:
  ```
  struct ListNode
  {
      int data;
      ListNode *next;
  };
  ```

- No memory is allocated at this time
Defining a Linked List

• Define a pointer for the head of the list:
  
  ```
  ListNode *head = NULL;
  ```

• Head pointer initialized to NULL to indicate an empty list
NULL Pointer

• Is used to indicate end-of-list
• Should always be tested for before using a pointer:

```c
ListNode *p;
while (p != NULL) ... 
```

• Can also test the pointer itself:

```c
while (!p) ... // same meaning
     // as above
```
17.2

Linked List Operations
Linked List Operations

• Basic operations:
  – append a node to the end of the list
  – insert a node within the list
  – traverse the linked list
  – delete a node
  – delete/destroy the list
Contents of NumberList.h

1  // Specification file for the NumberList class
2  #ifndef NUMBERLIST_H
3  #define NUMBERLIST_H
4
5  class NumberList
6  {
7     private:
8         // Declare a structure for the list
9         struct ListNode
10         {
11             double value;       // The value in this node
12             struct ListNode *next; // To point to the next node
13         };
14
15         ListNode *head;       // List head pointer
Contents of NumberList.h (Continued)

17    public:
18       // Constructor
19       NumberList()
20           { head = NULL; }  
21
22       // Destructor
23       ~NumberList();
24
25       // Linked list operations
26       void appendNode(double);
27       void insertNode(double);
28       void deleteNode(double);
29           void displayList() const;
30    
31    #endif
Create a New Node

• Allocate memory for the new node:
  
  ```c
  newNode = new ListNode;
  ```

• Initialize the contents of the node:
  
  ```c
  newNode->value = num;
  ```

• Set the pointer field to NULL:
  
  ```c
  newNode->next = NULL;
  ```
Appending a Node

• Add a node to the end of the list
• Basic process:
  – Create the new node (as already described)
  – Add node to the end of the list:
    • If list is empty, set head pointer to this node
    • Else,
      – traverse the list to the end
      – set pointer of last node to point to new node
Appending a Node

New node created, end of list located
Appending a Node

New node added to end of list
C++ code for Appending a Node

```cpp
void NumberList::appendNode(double num) {
    ListNode *newNode; // To point to a new node
    ListNode *nodePtr; // To move through the list

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

    // If there are no nodes in the list
    // make newNode the first node.
    if (!head)
      ...
```
C++ code for Appending a Node (Continued)

24    head = newNode;
25    else // Otherwise, insert newNode at end.
26    {
27        // Initialize nodePtr to head of list.
28        nodePtr = head;
29
30        // Find the last node in the list.
31        while (nodePtr->next)
32           nodePtr = nodePtr->next;
33
34        // Insert newNode as the last node.
35        nodePtr->next = newNode;
36    }
37  }
Program 17-1

// This program demonstrates a simple append
// operation on a linked list.
#include <iostream>
#include "NumberList.h"
using namespace std;

int main()
{
    // Define a NumberList object.
    NumberList list;

    // Append some values to the list.
    list.appendNode(2.5);
    list.appendNode(7.9);
    list.appendNode(12.6);
    return 0;
}

(This program displays no output.)
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

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

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 = NULL; // 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 = NULL;
    }
    else // Otherwise, insert newNode
    {
        // Position nodePtr at the head of list.
        nodePtr = head;
// Initialize previousNode to NULL.
previousNode = NULL;

// Skip all nodes whose value is less than num.
while (nodePtr != NULL && 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 == NULL) {

head = newNode;
newNode->next = nodePtr;
}
else // Otherwise insert after the previous node.
{
    previousNode->next = newNode;
    newNode->next = nodePtr;
}
}
Program 17-3

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
Traversing a Linked List

• Visit each node in a linked list: display contents, validate data, etc.

• Basic process:
  – set a pointer to the contents of the head pointer
  – while pointer is not NULL
    • process data
    • go to the next node by setting the pointer to the pointer field of the current node in the list
  – end while
Traversing a Linked List

`nodePtr` points to the node containing 5, then the node containing 13, then the node containing 19, then points to `NULL`, and the list traversal stops.
Deleting a Node

- Used to remove a node from a linked list
- If list uses dynamic memory, then delete node from memory
- Requires two pointers: one to locate the node to be deleted, one to point to the node before the node to be deleted
Deleting a Node

Locating the node containing 13
Deleting a Node

Adjusting pointer around the node to be deleted
Deleting a Node

Linked list after deleting the node containing 13
```cpp
void NumberList::deleteNode(double num)
{
    ListNode *nodePtr;       // To traverse the list
    ListNode *previousNode;  // To point to the previous node

    // If the list is empty, do nothing.
    if (!head)
        return;

    // Determine if the first node is the one.
    if (head->value == num)
    {
        nodePtr = head->next;
        delete head;
        head = nodePtr;
    }
    else
    {
```
{  
   // Initialize nodePtr to head of list
   nodePtr = head;

   // Skip all nodes whose value member is
   // not equal to num.
   while (nodePtr != NULL && nodePtr->value != num)
   {
      previousNode = nodePtr;
      nodePtr = nodePtr->next;
   }

   // If nodePtr is not at the end of the list,
   // link the previous node to the node after
   // nodePtr, then delete nodePtr.
   if (nodePtr)
   {
      previousNode->next = nodePtr->next;
      delete nodePtr;
   }
}
Program 17-4

// This program demonstrates the deleteNode member function.
#include <iostream>
#include "NumberList.h"
using namespace std;

int main()
{
    // Define a NumberList object.
    NumberList list;

    // Build the list with some values.
    list.appendNode(2.5);
    list.appendNode(7.9);
    list.appendNode(12.6);

    // Display the list.
    cout << "Here are the initial values: \n";
    list.displayList();
    cout << endl;
}
// Delete the middle node.
cout << "Now deleting the node in the middle.\n";
list.deleteNode(7.9);

// Display the list.
cout << "Here are the nodes left.\n";
list.displayList();
cout << endl;

// Delete the last node.
cout << "Now deleting the last node.\n";
list.deleteNode(12.6);

// Display the list.
cout << "Here are the nodes left.\n";
list.displayList();
cout << endl;
// Delete the only node left in the list.
cout << "Now deleting the only remaining node.\n";
list.deleteNode(2.5);

// Display the list.
cout << "Here are the nodes left.\n";
list.displayList();
return 0;

Program 17-4 (continued)

Program Output
Here are the initial values:
2.5
7.9
12.5

Now deleting the node in the middle.
Here are the nodes left.
2.5
12.5

Now deleting the last node.
Here are the nodes left.
2.5

Now deleting the only remaining node.
Here are the nodes left.
Destroying a Linked List

- Must remove all nodes used in the list
- To do this, use list traversal to visit each node
- For each node,
  - Unlink the node from the list
  - If the list uses dynamic memory, then free the node’s memory
- Set the list head to `NULL`
NumberList::~NumberList()
{
    ListNode *nodePtr; // To traverse the list
    ListNode *nextNode; // To point to the next node

    // Position nodePtr at the head of the list.
    nodePtr = head;

    // While nodePtr is not at the end of the list...
    while (nodePtr != NULL)
    {
        // Save a pointer to the next node.
        nextNode = nodePtr->next;

        // Delete the current node.
        delete nodePtr;

        // Position nodePtr at the next node.
        nodePtr = nextNode;
    }
}
17.3

A Linked List Template
A Linked List Template

• When declaring a linked list, must specify the type of data to be held in each node
• Using templates, can declare a linked list that can hold data type determined at list definition time
• See LinkedList.h (versions 1 and 2) and Program 17-5
17.4 Variations of the Linked List
Variations of the Linked List

- Other linked list organizations:
  
  - doubly-linked list: each node contains two pointers: one to the next node in the list, one to the previous node in the list
Variations of the Linked List

• Other linked list organizations:
  – circular linked list: the last node in the list points back to the first node in the list, not to NULL
The STL *list* Container
The STL list Container

- Template for a doubly linked list
- Member functions for
  - locating beginning, end of list: front, back, end
  - adding elements to the list: insert, merge, push_back, push_front
  - removing elements from the list: erase, pop_back, pop_front, unique
- See Table 17-1 for a list of member functions