Chapter 17: Linked Lists
ADT: ABSTRACT DATA TYPES

- Data structure - particular way of organizing data in a computer so that it can be used efficiently:
  - A university needs to manage a lot of individual students
  - A company has many clients
  - An RPG needs to manage many different characters
  - Working with individual objects is inefficient
  - Group objects together in a data structure

- What data structures do you know?
  - Arrays
  - STL vectors
Arrays

What’s wrong with arrays?

- Fixed size

Can you resize an array?

- Not directly
- Need to create another array of a different size:
  - Copy all elements from old array to new array
  - Delete the old array

What if you have 1 000 entries in your array, and you need to increase the size of array by 1?
Inserting Element into Array

What must I do now?
Inserting Element Into Array

5 2 7 -5 16 2

shift these elements one position right

18

want to insert 18 at position 2
Inserting Element Into Array

put 18 to the position 2
Deleting Element From Array

want to remove element at the position 3

Deleting Element From Array

want to remove element at the position 3

shift these elements one position left
Deleting Element From Array

What about vectors?

How can we solve this?

element has been successfully removed

5 2 7 16 2 ? ? ? ? ?
Introduction to the Linked List ADT

- **Linked list**: set of data structures (nodes) that contain references to other data structures
- **Every node has 2 components:**

  ![Diagram of a node with data members and pointer](image)
Introduction to the Linked List ADT

Called a linked list because every node has a pointer that points to next node in the list

Data members | Pointer
Introduction to the Linked List ADT

- List head is a pointer that points to first node of the list
- Each node points to next node in list
- Last node points to the null address
- Take note: nodes can be scattered around in memory

list head

null
Introduction to the Linked List ADT

- Have a list of linked nodes
- Nodes can be added to or removed from the linked list during execution
- Linked list only has to store the address of the first node (list head)
- Every node knows how to get to the next node
- Last node points to NULL
Linked Lists vs. Arrays and Vectors

- Linked lists can grow and shrink as needed, unlike arrays.
- Linked lists can insert a node between other nodes easily.

Only change pointers and we are done!
Node Organization

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

- Linked list contains 0 or more nodes:

  ![Diagram of a linked list with a list head pointing to first node, and the last node pointing to null](image)

  - list head

- Has a list head to point to first node
- Last node points to `null` (address 0)

How do we know that the linked list is empty?
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:

```c
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:
  ```cpp
  ListNode *head = nullptr;
  ```
- Head pointer initialized to `nullptr` to indicate an empty list
The Null Pointer

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

```c
ListNode *p;
//test whether the pointer is null
while (!p)
    //list is not empty and haven’t
    //reached end of list yet
```
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

Add node
Add node
Iterate through linked list
Remove node
NumberList: Linked List of Doubles

class NumberList
{
    private:
    struct ListNode // node structure
    {
        double value;  // The value in this node
        struct ListNode *next;  // Address of the next node
    };
    ListNode *head;  // List head pointer

    public:
    NumberList() { head = NULL; } // constructor: initialize head to 0
    ~NumberList(); // destructor

    // Linked list operations:
    void appendNode(double);  // add node at the end of the list
    void insertNode(double);  // insert a node (maintain ascending order)
    void deleteNode(double);  // remove a node
    void displayList() const; // print list on the screen
};
Create a New Node

When inserting a node, we first need to create the new node (4 steps required):

1. Allocate memory for the new node:
   
   ```
   newNode = new ListNode;
   ```

2. Initialize the contents of the node:
   
   ```
   newNode->value = num;
   ```
Create a New Node

When inserting a node, we first need to create the new node (4 steps required):

3. Set the pointer field to `nullptr`:

   ```cpp
   newNode->next = nullptr;
   ```

4. Set `list head` to `newNode`

   ```cpp
   head = newNode;
   ```
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
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;  // must we use "new"?
    newNode->value = num;  // set value to num
    newNode->next = NULL;  // set next to NULL

    // If there are no nodes in the list
    // make newNode the first node:
    if (!head)  // "If head is null"
        head = newNode;  // both are pointers
C++ Code To Append A Node

```cpp
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) { // “while next != 0”
32             nodePtr = nodePtr->next;
33         }
34         // Insert newNode as the last node
35         nodePtr->next = newNode;
36     }
37 }
```
Program 17-1

1 // This program demonstrates a simple append
2 // operation on a linked list.
3 #include <iostream>
4 #include "NumberList.h"
5 using namespace std;
6
7 int main()
8 {
9     // Define a NumberList object.
10    NumberList list;
11
12    // Append some values to the list.
13    list.appendNode(2.5);
14    list.appendNode(7.9);
15    list.appendNode(12.6);
16    return 0;
17 }

(This program displays no output.)