Linked List

**Overview**

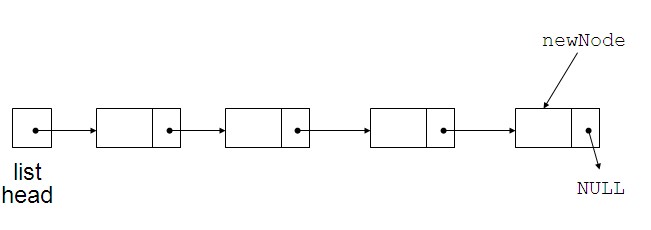
Second most basic of dynamic data structures - most basic is dynamic arrays. Needed when:

1. Don’t know how many elements are to be stored

2. Need flexibility when removing or inserting elements

3. Unlike arrays, don’t have capacity.

Linked Lists are created by *Nodes* dynamically allocated in memory. The memory is not contiguous, and must be *linked* together as shown.

Has a list head to point to first node

Last node points to NULL

NULL

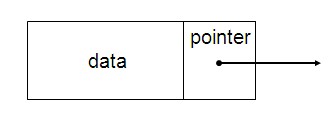
list

head

**Can Insert Nodes Anywhere Easily**

A node contains:

* + data: one or more data fields – may be organized as structure, object, etc.
  + a pointer that can point to another node



An example declaration of this node is shown below:

struct Node

{

int data;

Node \*next;

};

Basic operations:

* + append a node to the end of the list
  + insert a node within the list
  + traverse the linked list
  + delete a node

**Linked List Class**

Basic Class Declaration for Linked List is given below for GenPoint:

struct Node

{

GenPoint data;

Node\* next;

};

class LinkedList

{

private:

Node \*first; //Points to start of the list

public:

LinkedList(); // Constructor.

~LinkedList(); // Destructor.

LinkedList(const LinkedList& list); //Copy Constructor

…other methods follow

};

**Constructor sets *first* to NULL:**

LinkedList::LinkedList()

{

first = NULL;

}

**Destructor must delete one Node at a time:**

LinkedList::~LinkedList()

{

Node\* temp;

Node\* current;

current = first;

//Clean up all nodes by walking the list

while(current)

{

//Save off the current node

temp = current;

//Go to next node

current = current->next;

//Cleanup

delete temp;

}

//Init first back to NULL for empty list

first = NULL;

}

**Appending a Node**

void LinkedList::addNode(GenPoint point)

{

Node\* current;

Node\* new\_node = new Node;

new\_node->point.setPoint(point.getX(), point.getY());

new\_node->next = NULL;

//Determine if empty list

if (!first)

{

first = new\_node;

return;

}

//OW Locate last node by walking the list

current = first;

while(current->next)

{

current = current->next;

}

//Add the node here

current->next = new\_node;

}

**Copy Constructor**

LinkedList::LinkedList(const LinkedList& list)

{

Node\* current; // points to each node in sequence

first = NULL; // Initialize first before calling addNode()

current = list.first; // Initialize current to the first node parameter

// Append new nodes to this list that are copies of the list’s nodes

while (current)

{

addNode(current->point);

current = current->next;

}

}

**Remove a node using the following algorithm:**

bool LinkedList::removeNodeAt (int index)

{

Node \*node\_ptr;

Node \*delete\_ptr;

int counter;

//Check for an empty list

if (index < 0)

{

return(false);

}

//Check for index >= number of elements

if (index >= getNoOfElements() ) //What does getNoOfElements involve?

{

return(false);

}

//Check for index == 0

if (index == 0)

{

delete\_ptr = first;

first = first->next;

delete delete\_ptr;

return(true);

}

//Case for valid index

node\_ptr = first;

for (counter = 0; counter < index - 1; counter++)

{

node\_ptr = node\_ptr->next;

}

delete\_ptr = node\_ptr->next

node\_ptr->next = node\_ptr->next ->next;

delete delete\_ptr;

return true;

}