The Problem with Arrays

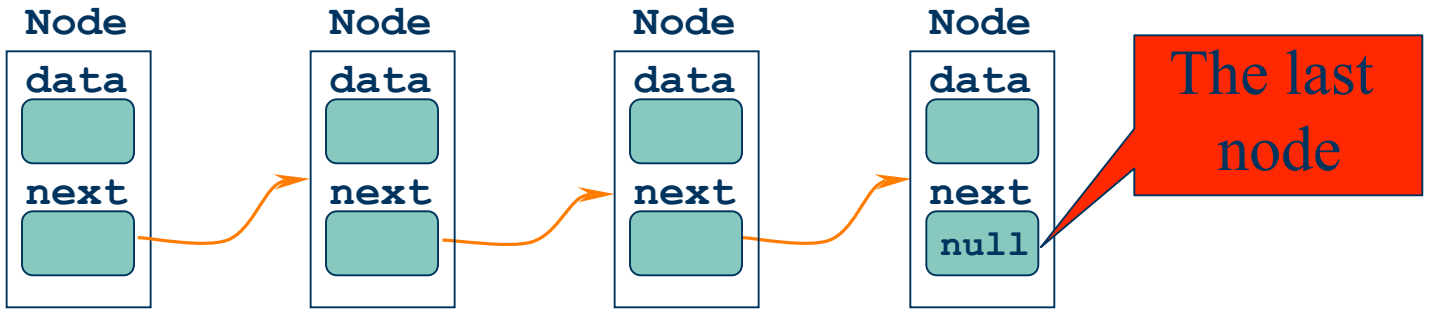
- Fixed, static size – once created, can not be changed, at least not easily.
- Because of static size, often a much bigger than needed size is allocated just to be "safe". (e.g. doubling of stacks and queues)
- Deletion from or insertion into the middle of the array is costly – must shift the rest of the array.

The Linked List

- The second most commonly used data structure, after arrays.
  - If you count objects as a data structure, then third
- Superior to arrays!!!!
  - Except access via indexing is used frequently.
  - Also, if space is known arrays are more efficient
- Many variations

Links and References

- In a linked list, each data object is embedded in a node.
- In each node, in addition to the data, there is also a reference to the “next” node.
- The reference is what keeps the list structure.



Node class

```
class Node {
  int idata;        // data
  double ddata;     // data
  Node next;        // reference
}
```

- Note that the reference next refers to an object of the same type as itself.

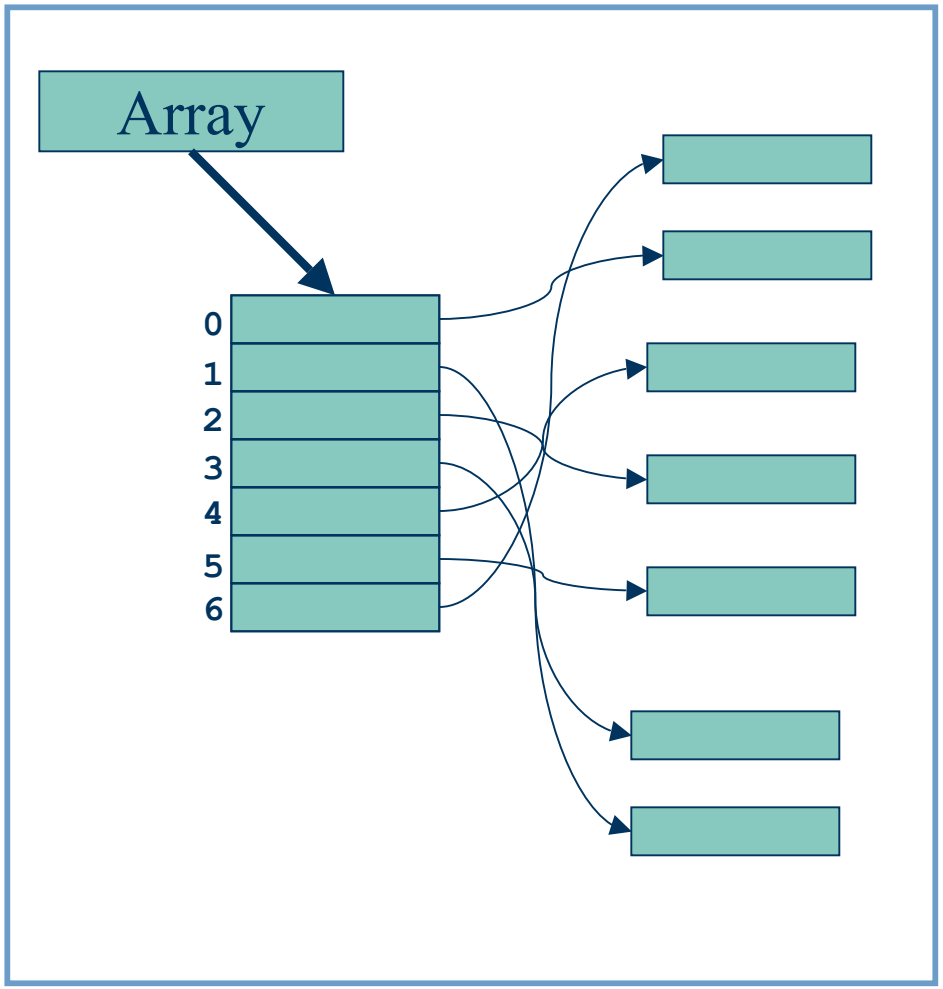
References

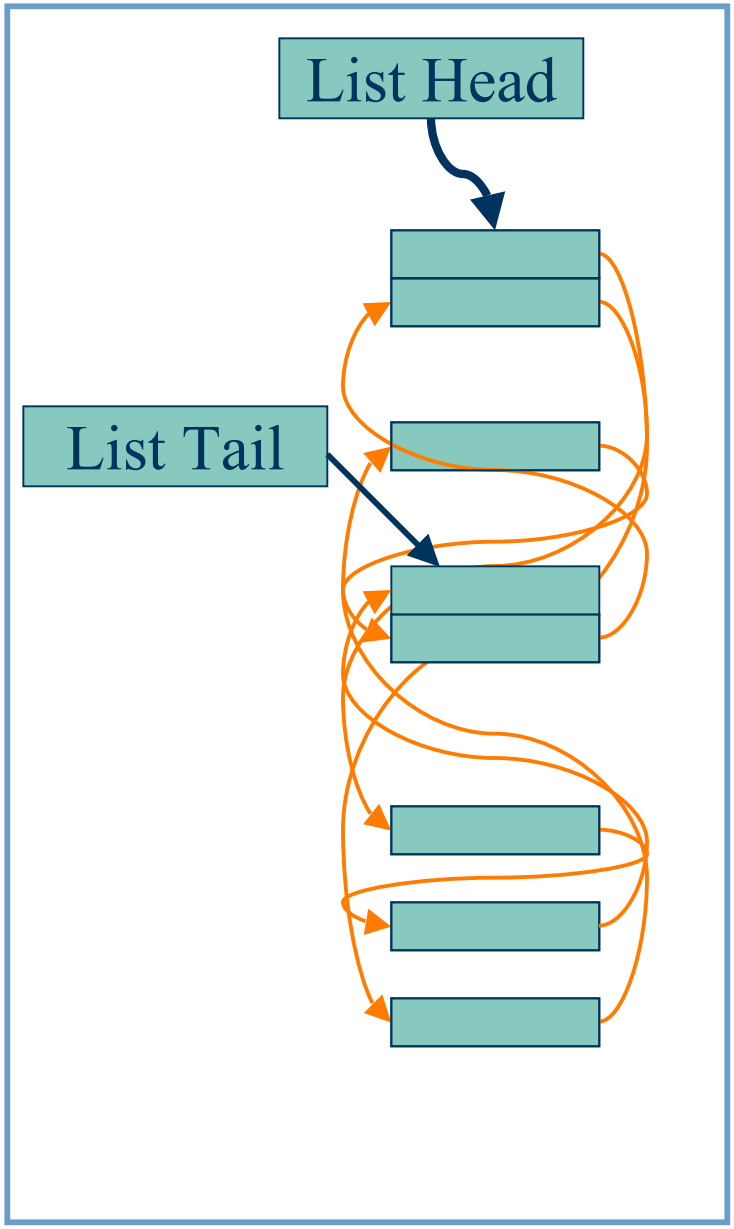
- Remember that references are variables that refers to another object.
- References store the memory address of the actual object it refers (points) to.
- All references are of the same size on a particular machine.

Arrays -- Position
Linked Lists -- Relationship

- In an array, we go to the next data item by increasing the index by 1.
- All array elements (which are only references) are stored sequentially in memory –
  - but the objects may be scatterted.
- In a linked list, we follow the reference to go to the next data item.
- The actual nodes may be scattered all over in memory.

Array and Linked List in Memory





Simple Linked List

first define the “node”

```
public class Node {
   private int idata;              // data
   private Node next;              // reference
   public Node(int x) {
       idata = x; next = null;
   }
   public String toString() {
       return idata+"";
   }
   public void setNext(Node next) {
       this.next=next;    }
   public Node getNext() { return next; }
   public int getIdata() { return idata; }
}
```

Simple Linked List

LinkedList Class

```
public class LinkedList {
  private Node head, tail;
  public void LinkedList() {
     head = null;
     tail = null;}
  public boolean isEmpty(){
     return (head == null);}
  public Node first(){return head;}
  public Node last(){return tail;}
   ...
```

`append()` method

```
public void append(Node n) {
   if (isEmpty()) {
     head = tail = n;
   }
   else {
     tail.next = n;
     tail = n;
   }
}
```

`prepend()` method

```
public void prepend(Node n) {
   if (isEmpty()) {
     head = tail = n;
   }
   else {
     n.next = head;
     head = n;
   }
}
```

`printList()` method

```
public void printList()
    {
       System.out.println(l.toString());
    }

    public String toString()
    {
       String s = "";
       if (isEmpty())
              return "List is empty";
       for(Node tmp=head;tmp!=null;tmp=tmp.getNext())
              s = s + tmp+" ";
       return s;
    }
```

main

```
class LinkedListApp {
  public static void main(String[] args){
    LinkedList l = new LinkedList();
    for (int i=0; i<10; i++) {
      Node newnode = new Node(i);
      l.append(newnode);}
    for (int i=10; i<20; i++) {
      Node newnode = new Node(i);
      l.prepend(newnode); }
    l.printList();
  }
}
```

`find()` method

```
public Node find(int key) {
    Node tmp;
    for(tmp=head;tmp!=null;tmp=tmp.getNext())
      if(tmp.getIdata() == key)
          return tmp;
    return null;
}
```

`delete()` method

```
public void delete(Node n) {
   Node tmp, prev=null;
   if (n == head && n == tail) {
       head = tail = null; }
   else if (n == head) {
       head = n.getNext();; }
   else {
       for(tmp=head;tmp!=null;prev=tmp,tmp=tmp.getNext())
       if(tmp == n)
           break;
       prev.setNext(tmp.getNext());
       if (tmp == tail)
       tail = prev;
     }}
```

main (more)

```
    ...
    l.delete(l.first());
    l.printList();
    l.delete(l.find(3));
    l.printList();
    l.delete(l.find(10));
    l.printList();
    l.delete(l.last());
    l.printList();

  for (int i=0; i<20; i++) {
      Node n = l.find(i);
      if (n != null)
        l.delete(n);
    }
    l.printList();
```