ZAMAN UNIVERSITY

Data Structures and Algorithms

Chapter 2

Abstract Data Types

Outline

- Stacks
- Queues and Priority Queues
- Linked Lists
- Abstract Data Types
- Specialized Lists

Outline

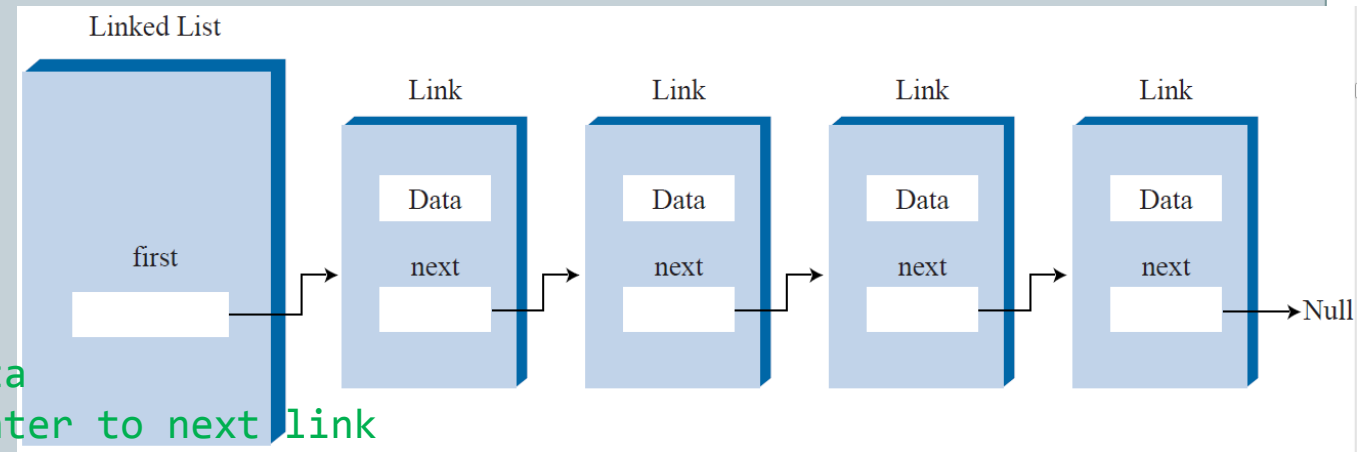
- Stacks
- Queues and Priority Queues
- Linked Lists
- Abstract Data Types
- Specialized Lists

Linked Lists

- In an unordered array, searching is slow, whereas in an ordered array, insertion is slow. In the both kinds, deletion is slow
- Also, the size of array cannot be changed after it is created
- The Linked List solves some of these problems
- Linked Lists are probably the second most commonly used general-purpose storage structures after array

Understanding Links

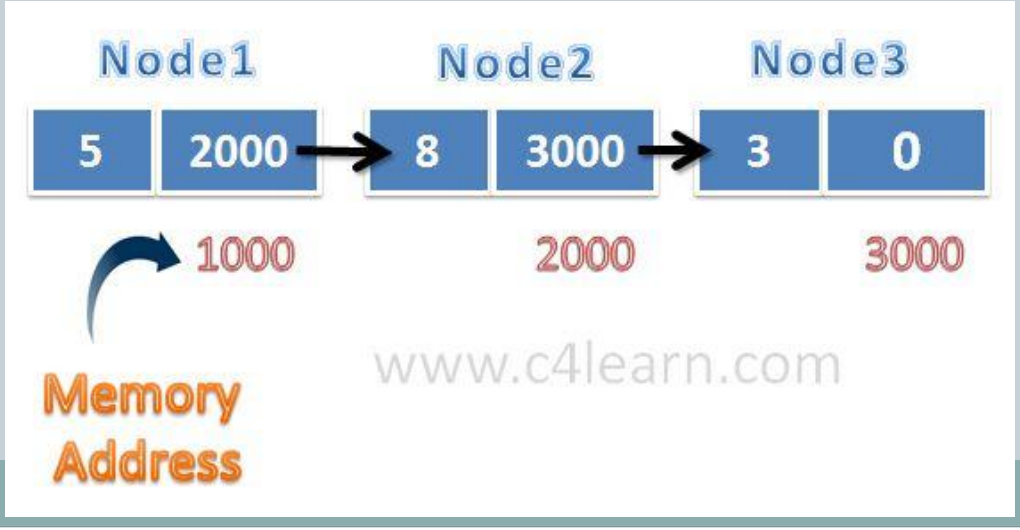
- In a linked list, each data item is embedded in a link
- A *link* is an object of a class called something like **Link**
- Because there are many similar links in a list, it makes sense to use a separate class for them, distinct from the linked list itself
- Each link object contains a pointer (which we'll call **pNext**) to the next link in the list.




```
class Link
{ public:
   int iData; //data
   double dData; //data
   Link* pNext; //pointer to next link
};
```

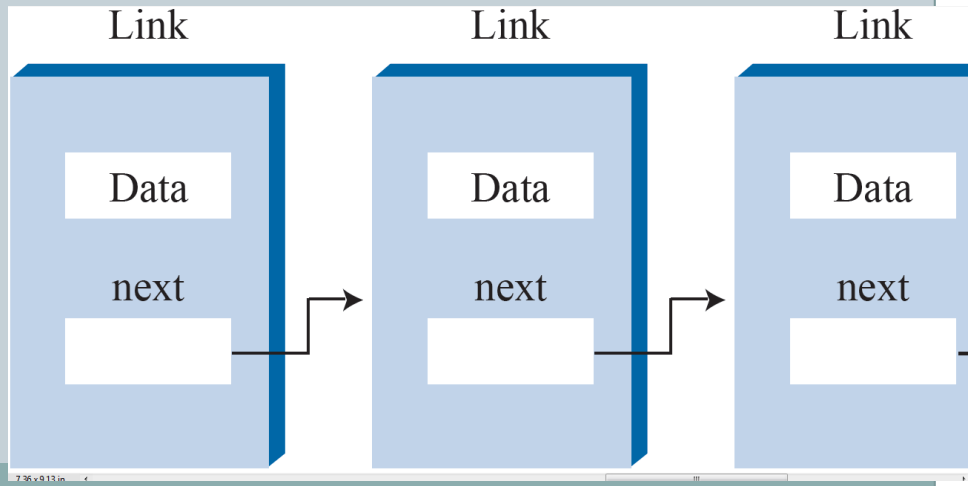
Linked Lists: Structure Defined by Relationship, Not Position

- In an array each item can be directly accessed using an index number
- In a list the only way to find a particular element is to follow along the chain of links
- You can't access a data item directly; you must use relationships between the items to locate it
- You start with the first item, go to the second, and then the third, and so on, until you find what you're looking for.



Create A Link in C++

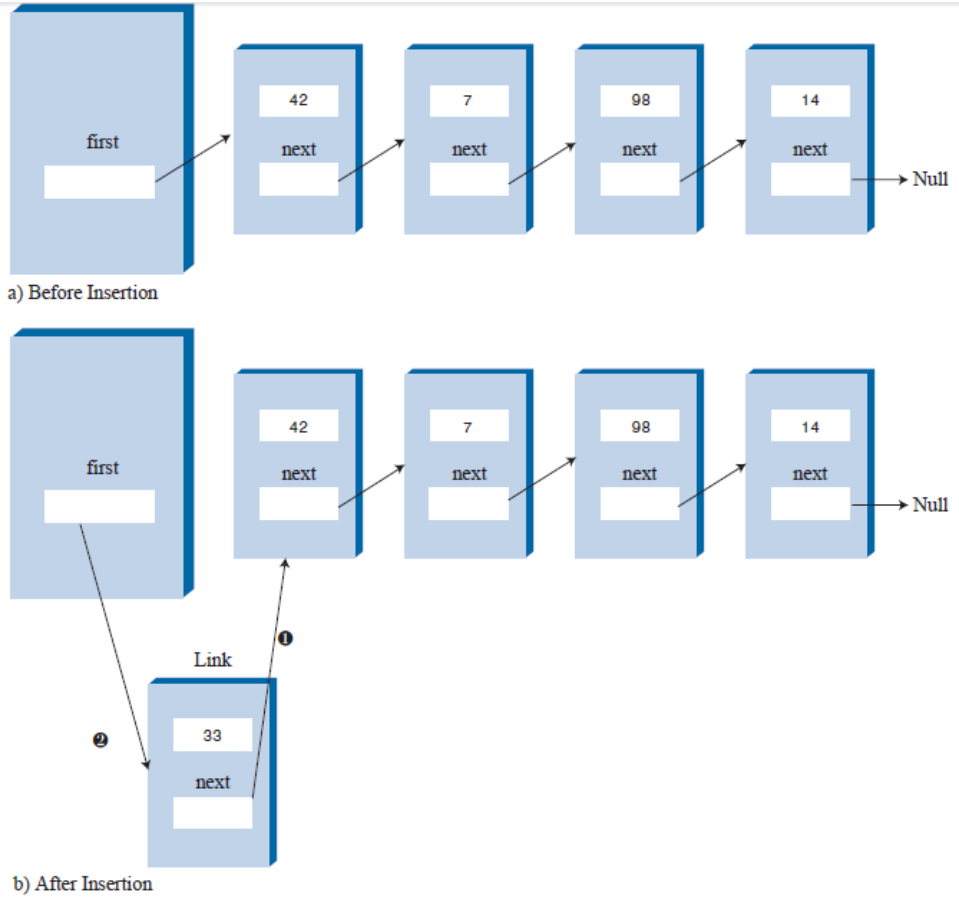

```
class Link{
  public:
    int iData;                      //Data Item
    string sName;                   //Data Item
    Link* pNext;                    //Pointer to the next link in list

    void InitLink(int id, string sn);       //Initiation Function to class attributes
    void DisplayLink();             //Display value of Items
};

void Link::InitLink (int id, string sn){
    iData = id;
    sName = sn;
    pNext = NULL;
};
void Link::DisplayLink(){
    cout << " {iData: " << iData;
    cout << " sName: " << sName << "}" << endl;
};
```



Create A Linked List in C++

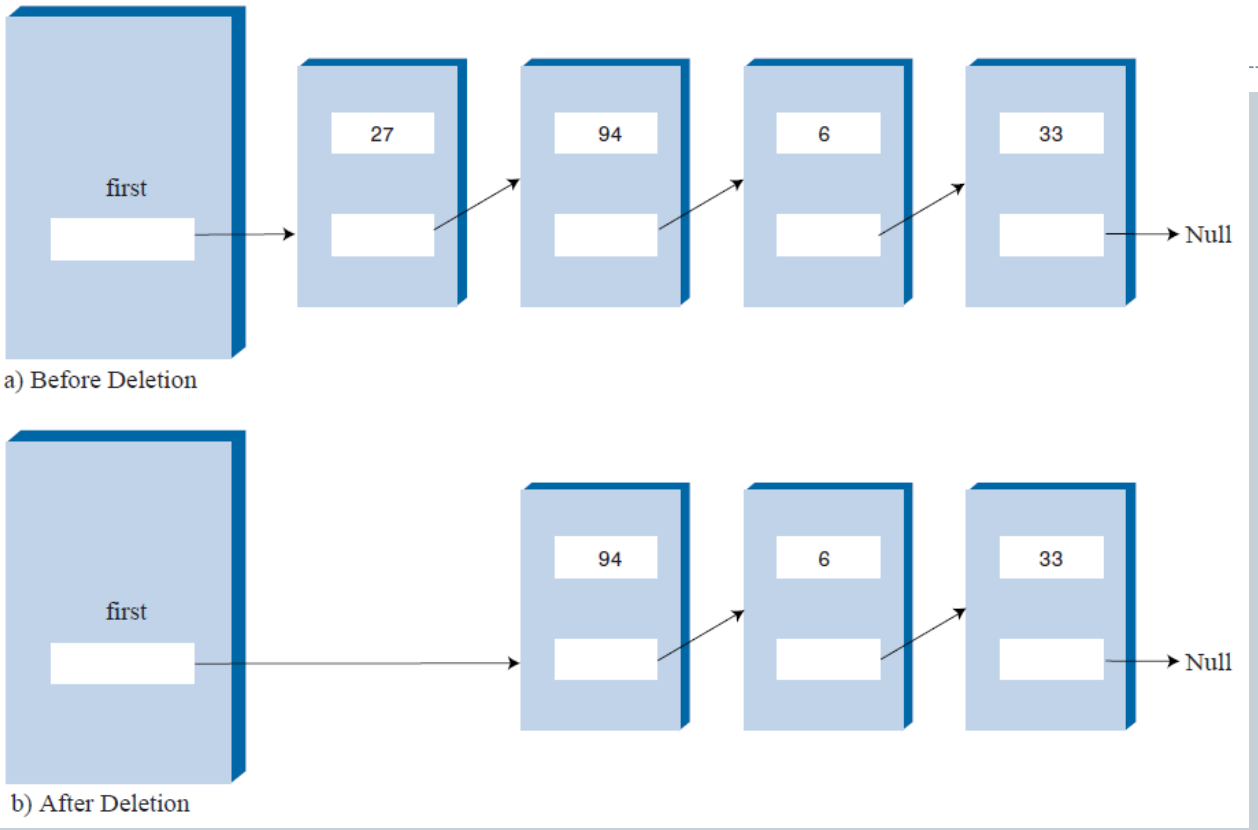

```
class LinkList{
    Link* pFirst;          //ptr to first link on list
public:
    void InitLinkList();
    bool IsEmpty();
    void InsertFirst(int d, string s);
    void RemoveFirst();
    void DeleteLinkList();
    //... Other Methods here
};

void LinkList::InitLinkList(){
    pFirst = NULL;                          //(no links on list yet)
};

bool LinkList::IsEmpty(){
    if(pFirst == NULL ) return true;
    else return false;
};
```

The *InsertFirst()* Function Member




```
void LinkList::InsertFirst(int i, string s){
    Link* NewLink = new Link();       //make new link
    NewLink->InitLink(i, s);
    NewLink->pNext = pFirst;          //newLink-->old first
    pFirst = NewLink;                 //first-->newLink
};
```

The RemoveFirst() Member Function




```
void removeFirst()                  //delete first link
{ //(assumes list not empty)
        Link* pTemp = pFirst;       //save first
        pFirst = pFirst->pNext; //unlink it: first-->old next
        delete pTemp;               //delete old first
}
```

Delete LinkList (Clear LinkList from Memory)


```
void DeleteLinkList() {
    Link* pCurrent = pFirst;          //start at beginning of list
    while(pCurrent != NULL) {         //until end of list,
        Link* pOldCur = pCurrent;     //save current link
        pCurrent = pCurrent->pNext;   //move to next link
        delete pOldCur;               //delete old current
    }
}
```

Create the following function members:

1. void Size() – return number item/link in the list;
2. void InsertToPosition(int intData, string strData, int pos) – Create a new link with these values: i & s. In case pos is in the range thus insert the new link to the position;
2. void InsertTail(int i, string s) – insert a link with value i and s to tail of the list;
3. void RemoveTail() – remove a link from the list;
4. Link* GetTail() – return a link at tail of the list;
5. Link* FirstTail() – return the first link or head link;
6. Link* Find(int data) – return a link with a value iData equal to data;

Read book of **Robert Lafore**, page: 146– 165 for next lecture

To be continued…