Data structure and Algorithms

Presented by : Asmaa Ghonaim

Lists

Lists: Very useful data structure.

- Are a way to store many different values under a single variable.
- Every item Node in this list is numbered with an index.
- Unlike counting things that exist in the real world, index variables always begin with the number 0.

Lists

List Operations:

- **Add** : add a new node
- **Set** : update the contents of a node
- **Remove**: remove a node
- **ISEmpty** : reports whether the list is empty
- **ISFull** : reports whether the list is full
- **Initialize** : create/initialize the list
- **Destroy** : delete the content of the list.

Lists

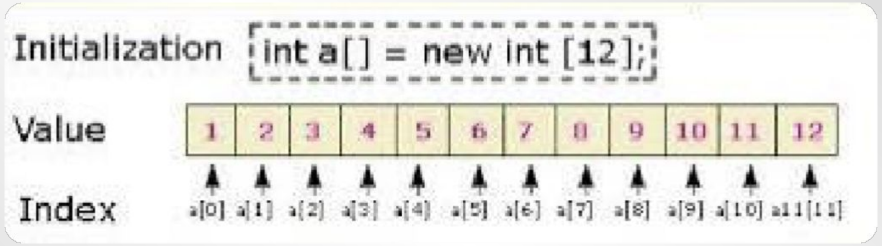
Lists Types:

Arraylist :

- Allocate the memory for all its elements in one block of memory.
- The size of the array is fixed.

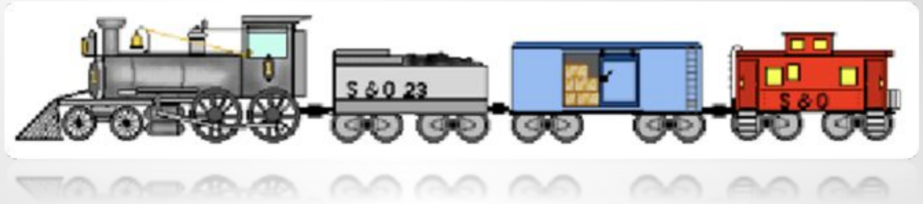
```
        Person P[10];
        Person *p;
// also here you need to know the length early

        P= new Person[10];
```

The size of the array is fixed.

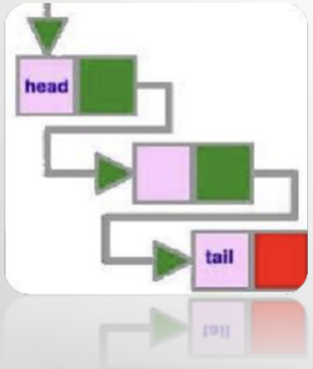


```
int a[] = new int [12];
```

Linked List



Linked List



Purposes of Linked Lists:

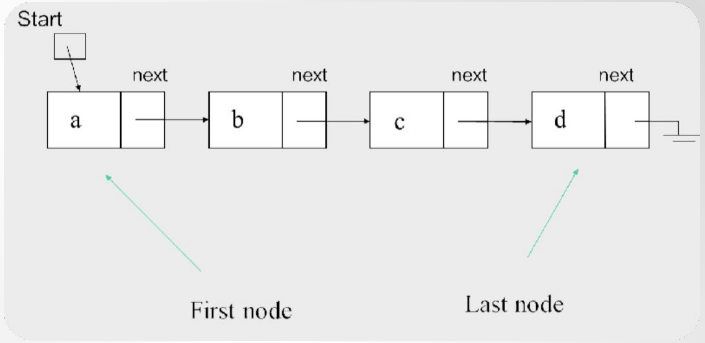
- To create arrays of unknown size in memory.
- Linked lists are dynamic, so the length of a list can increase or decrease as necessary.
- Each node does not necessarily follow the previous one physically in the memory.
- Linked lists can be maintained in sorted order by inserting or deleting an element at the proper point in the list.
- To store data in disk-file storage of databases.
    i. The linked list allows you to insert and delete items quickly and easily without rearranging- the entire disk file. For these reasons, linked lists are used extensively in database managers.

Linked List

Type of Linked Lists:

Single Linked list:

- A singly linked list contains a link to the next data node.
- Each node contains both its data and pointer to next node in the list.
- Need only one pointer refers to the beginning of the list.
- It is difficult to go back and move through the list.

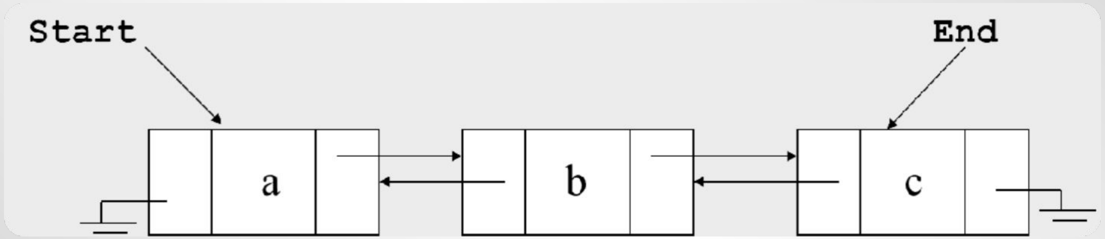


Linked List

Type of Linked Lists:

Doubly-Linked List:

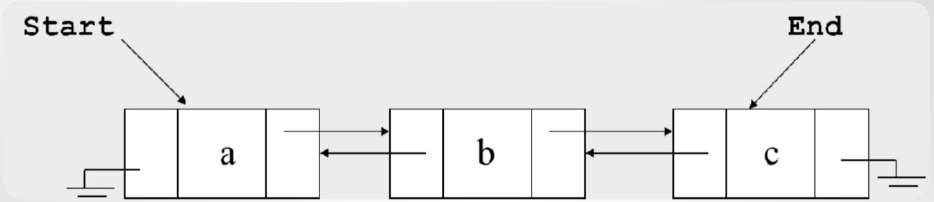
- Contains data and links to both next and the previous node in the list.
- Has pointer to the first node and another for the last node.
- The list can be read in either direction that simplifies sorting the list.
- Because either a forward link or a backward link can read the entire list, if one link becomes invalid, the list can be reconstructed using the other link. This is meaningful only in the case of equipment failure.



Linked List

Operations of Doubly-Linked List:

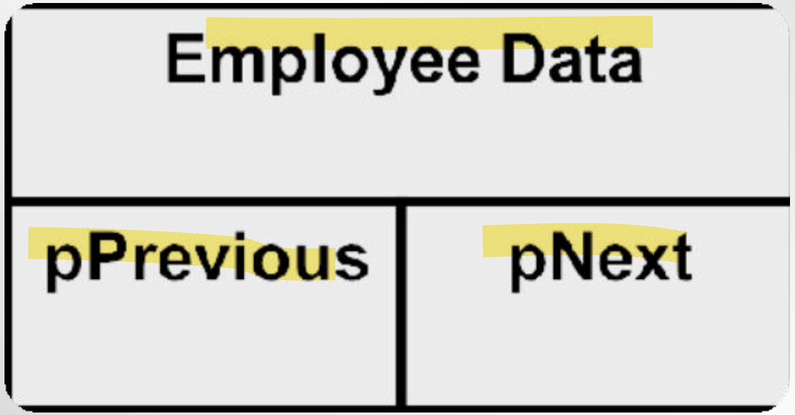
- **Inserting New Nodes**
  1. Case 1: insert new first node.
  2. Case 2: insert a new middle node.
  3. Case 3: insert a new last node.
- **Deleting Nodes**
  1. Case 1: delete a node.
  2. Case 2: deleting all nodes (removing the list).
- **Searching for a particular Node.**
- **Display All**



Linked List

The following example shows how to build a Doubly-Linked List, using nodes of Employee Structure.

- **Node Element Definition:**



Linked List

```
class Employee {
char name[20];
float salary;
float oTime;
 int code;
public:
   Employee *pNext; Employee *pPervious;
   Employee() { code = o;
                    name=”no name” ;
                    salary = oTime = 0;
                    pNext=NULL; pPrevious =NULL;}
Employee(int c, char *n, float s, float o) { code = c;
                         name= n ;
                         salary =s ; oTime =o;
                       pNext=NULL; pPrevious =NULL;}
// setters and getters.
void printEmployee();  };
```

Linked List

List Definition:

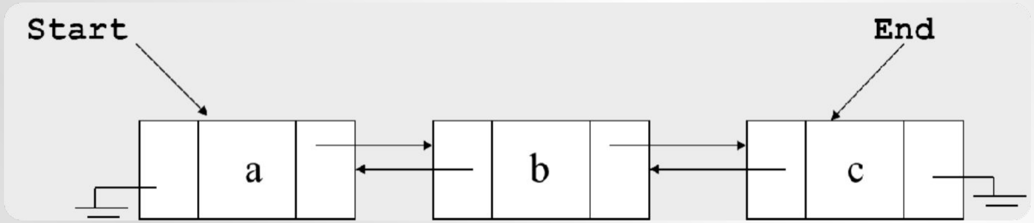
```
class LinkedList  {

protected:

Employee *pStart; Employee *pEnd;
public:
     LinkedList() {pStart=pEnd=NULL;}
// Setters and getters for pStart and pEnd;
    void addList(Employee *pItem);
    void InsertList(Employee *pItem);
    Employee* searchList(int Code)
   int DeleteList(int Code);
    void freeList();
     void displayAll();
   ~LinkedList() { freeList();}
};
```

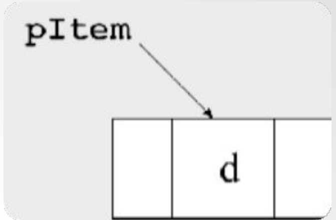
Doubly-Linked List Operations

Adding New Nodes

- **Case 1:** insert new first node.
- **Case 2:** insert a new last node.

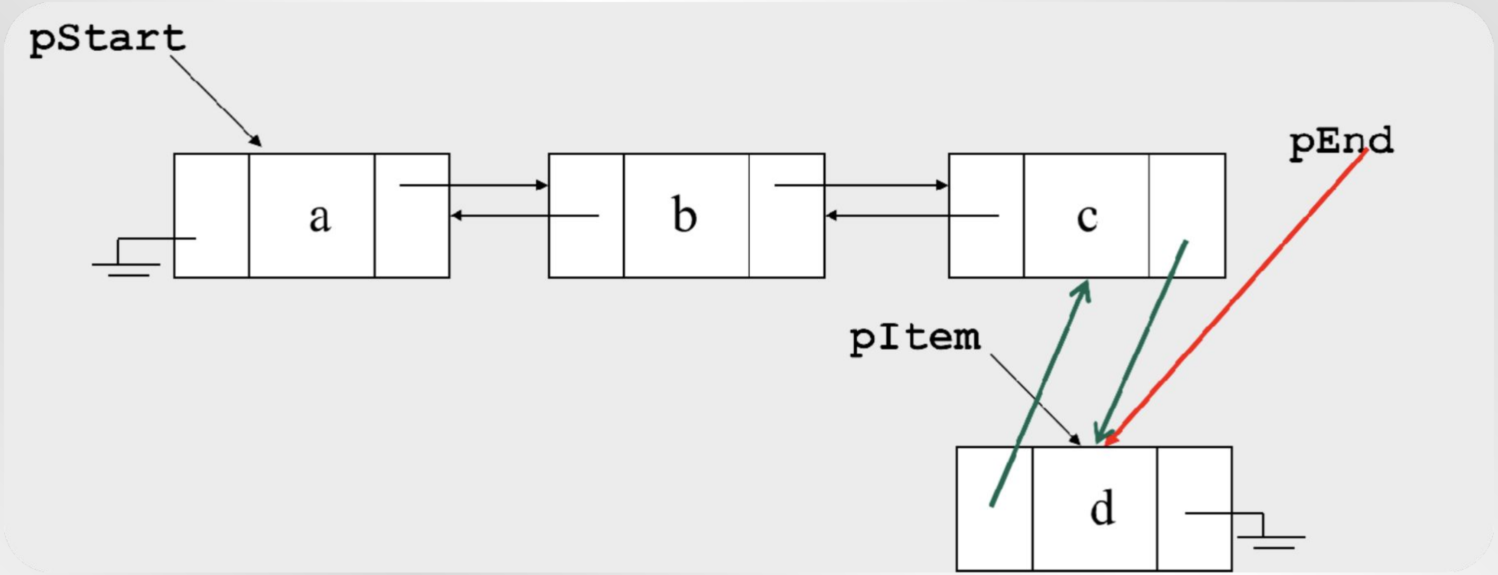
the addList() function in class LinkedList builds a doubly linked list. This function places each new node on the end of the list (i.e., append the node to the end of the list).





Doubly-Linked List Operations

Adding New Nodes



Doubly-Linked List Operations

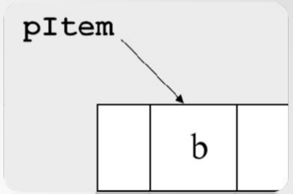
Adding New Nodes

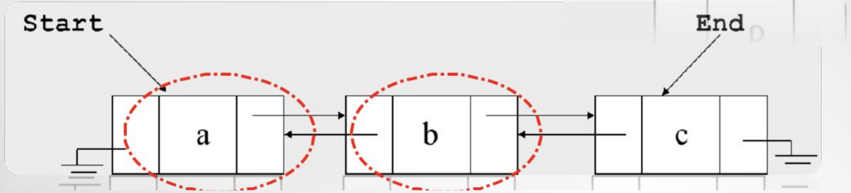
```
void LinkedList :: addList(Employee *pItem) {
    if (!pStart)                         /* Case of empty list Add the first node */
     {
       pItem->pNext = NULL;
       pItem->pPrevious = NULL;
       pStart = pEnd = pItem;
     }
   else                                  /* Case of existing list nodes Append Node */
    {
       pEnd->pNext = pItem;
       pItem->pPrevious = pEnd;
      pItem->pNext = NULL;
      pEnd = pItem;
     }
}
```

Doubly-Linked List Operations

Searching for a specified Node in the list

```
Employee* LinkedList :: searchList(int key)    // sequential search
{
    Employee * pItem= NULL;
    pItem = pStart;                       // begin from the first node
    while(pItem && pItem->getCode() != key) {
      pItem = pItem->pNext;
     }
   return pItem;
}
```

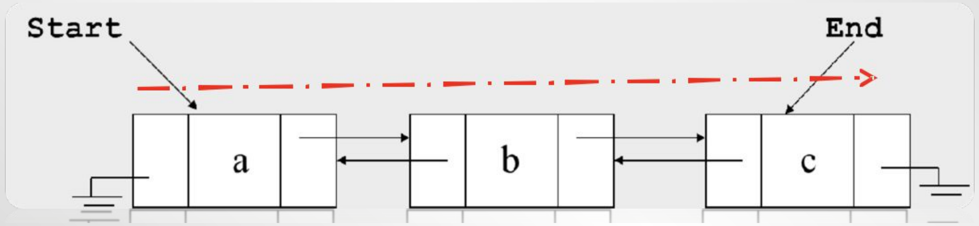




Doubly-Linked List Operations

Display All

```
void LinkedList :: displayAll()
{
    Employee * pItem;
    pItem = pStart;
    while(pItem)
  {
    pItem->printEmployee();
    pItem = pItem ->pNext;
  }
}
```



Doubly-Linked List Operations

Deleting a Node from the List

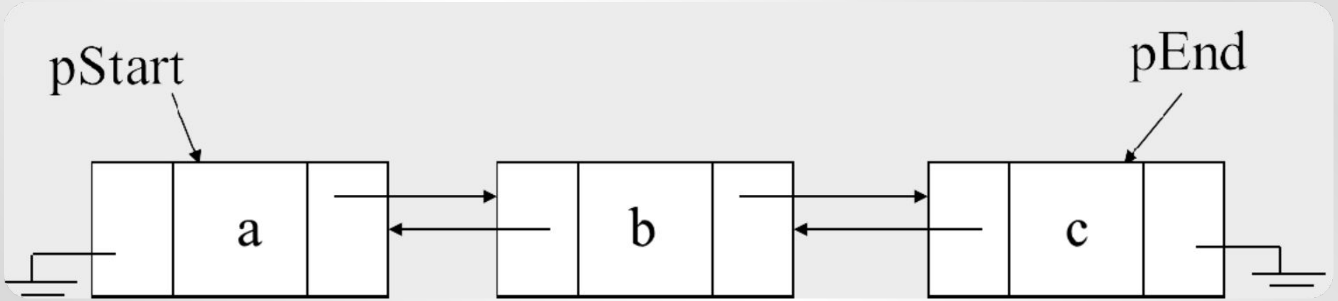
- Case 1: Deleting the first node.
- Case 2: Deleting a middle node.
- Case 3: Deleting the last node.

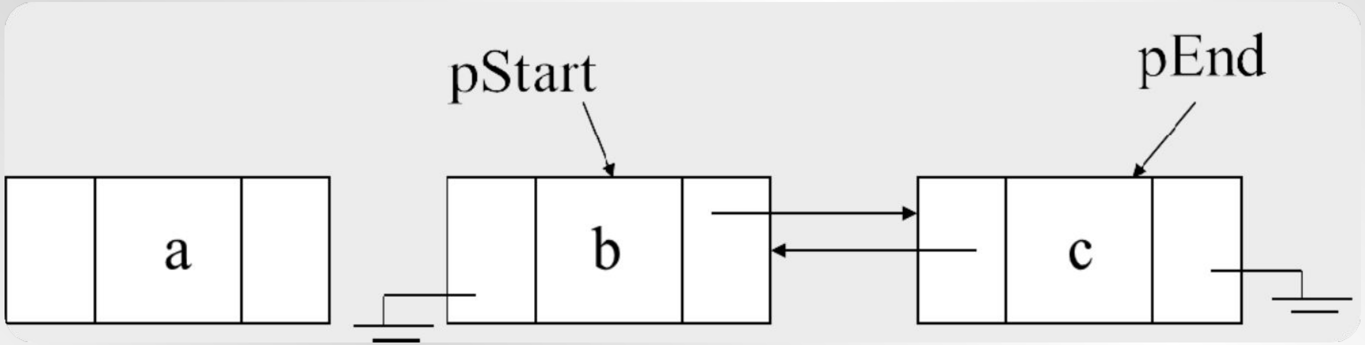
Steps:

- ❖ Search for the node first.
- ❖ No action, if it is not found.
- ❖ Found at the first place.
- ❖ Found at the last place.
- ❖ Found in middle.
- ❖ If it is the only node.

Doubly-Linked List Operations

Deleting a Node from the List

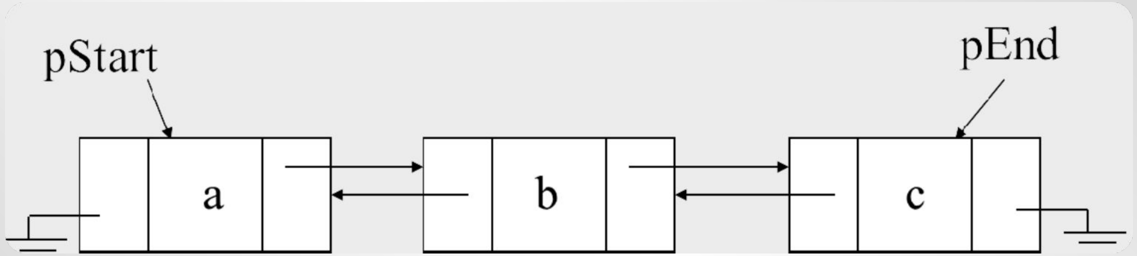


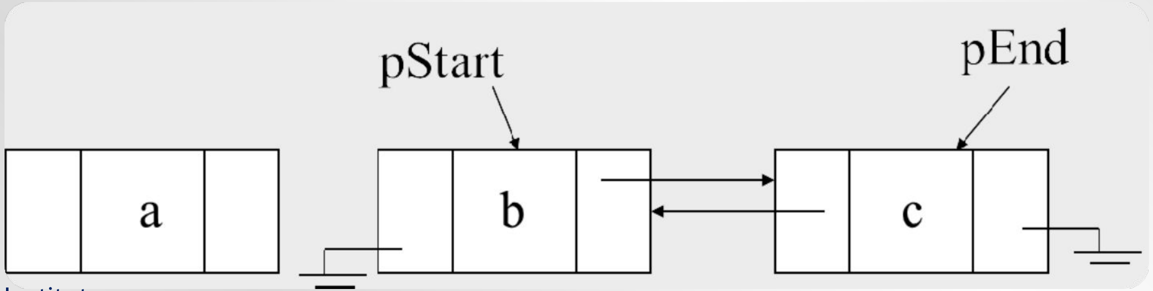


Doubly-Linked List Operations

Deleting a Node from the List

Case 1: Deleting the first node.

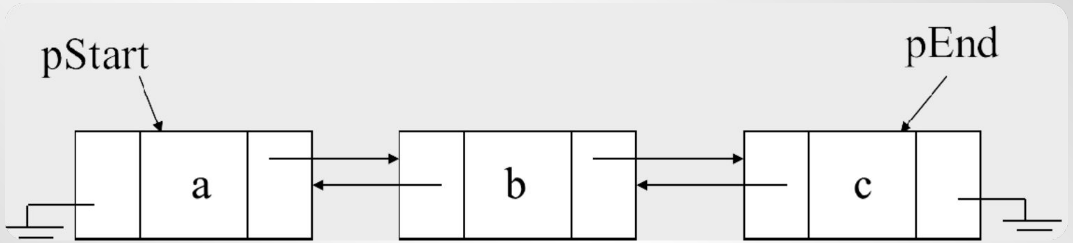


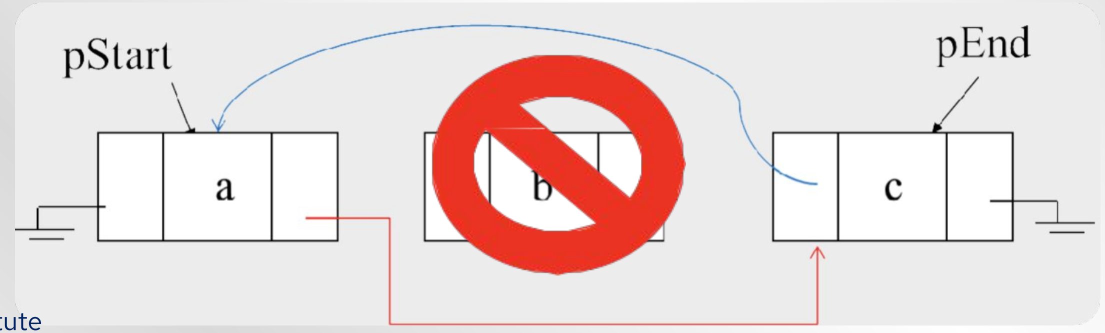


Doubly-Linked List Operations

Deleting a Node from the List

Case 2: Deleting a middle node.

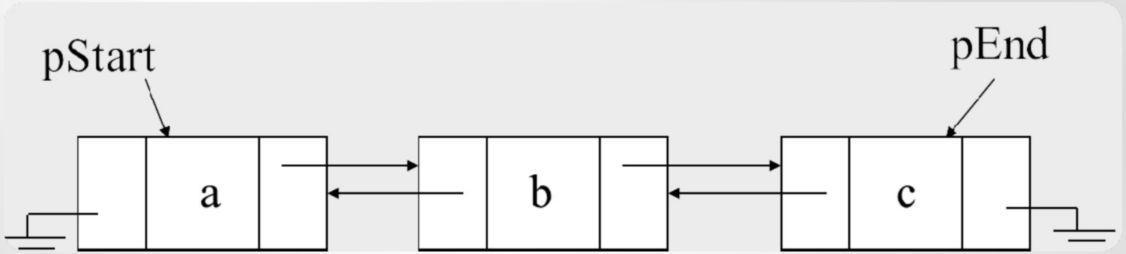


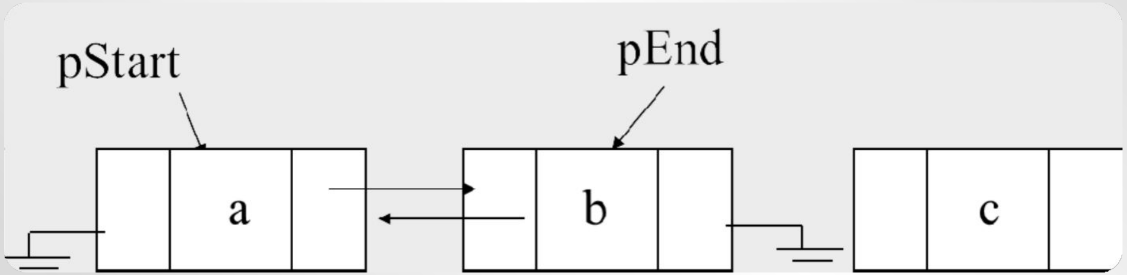


Doubly-Linked List Operations

Deleting a Node from the List

Case 3: Deleting the last node.





Doubly-Linked List Operations

Deleting a Node from the List

```
int LinkedList :: deleteList(int Code){
Employee *pItem;
int RetFlag = 1;
pItem = SearchList(Code);          // First search for it
if (!pItem) { RetFlag = 0; }         //Not found
else {                               // Found
     if (pStart == pEnd) {           /* Case of one node */
       pStart = pEnd = NULL; }
else if(pItem == pStart) {           /* Case of first node */
   pStart = pStart->pNext;
   pStart->pPrevious = NULL; }
else if(pItem == pEnd)  {            /* Case of last node */
   pEnd = pEnd->pPrevious;
   pEnd->pNext = NULL;  }
```

Doubly-Linked List Operations

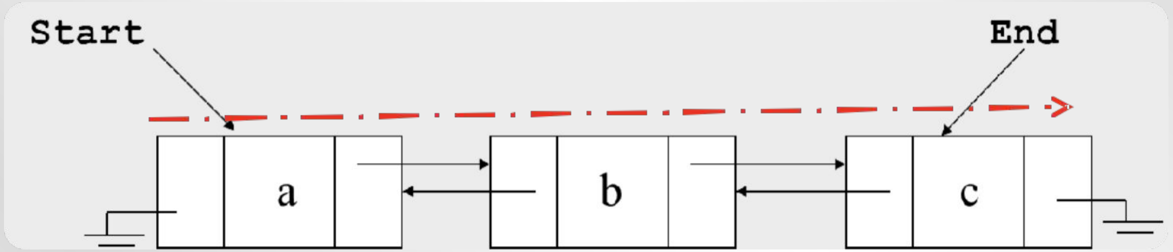
Deleting a Node from the List

```
else            /* Case of intermediate node */
   {
       pItem->pPrevious->pNext = pItem->pNext;
       pItem->pNext->pPrevious = pItem->pPrevious;
   }
      delete pItem;
 }
    return RetFlag;
}
```

Doubly-Linked List Operations

Removing the Entire List

```
void LinkedList :: freeList() {
       Employee * pItem;
while(pStart) {
      pItem = pStart;
      pStart = pStart->pNext;
      delete pItem;
   }
     pEnd = NULL;
}
```



Lab

Exercise

Assignments :

- ➔ Implement insert Node from the Doubly Linked list with all Cases
- ➔ **Bonus:** Sort Doubly Linked List.
- ➔ **Search:** Implement Circular Doubly Linked Lists operations.

THANKS!