Linked Lists

COMP 1002/1402

Using Defined Types

Data Structures are key to Computer Science

Think of the JAVA libraries:

- Vector,
- ArrayList,
- Hashtable…

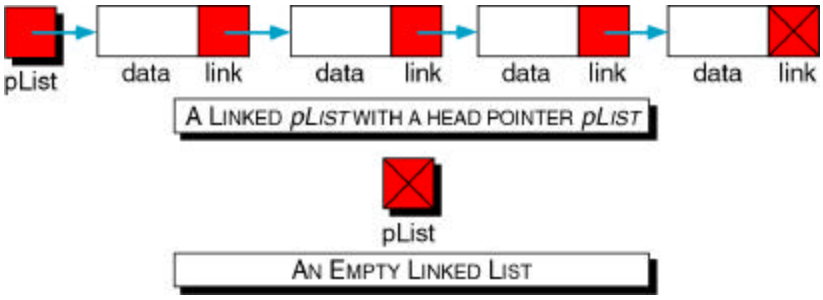
Lists of Information

Begin with a basic concept: A List

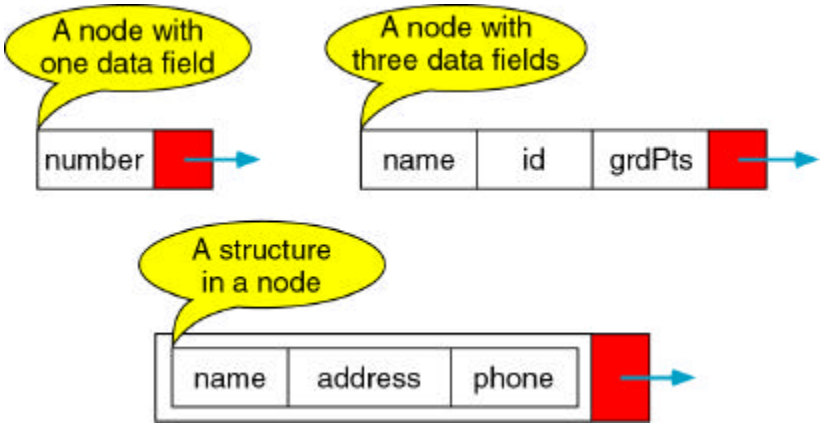
Has a first element
second element
third element…

Could be arbitrary length

Singly Linked Lists



Single link per element



So what is in a NODE?

Each element (NODE) in a list contains:

Data

and a pointer to another NODE

How will we define the NODE type?

Global Declarations!

```
typedef int KEY_TYPE; /*applic dependent*/

typedef struct {
  KEY_TYPE key;
  …  /* other data */
} DATA;

typedef struct {
  DATA data;
  struct nodeTag *link;
} NODE;
```

List Orderings

Three main types of lists:

- First in First out (queue)
- Last in First out (stack)
- Key Sequenced (sorted list)

Keeping a List

Keep a pointer to the first NODE

```
NODE * pList;
```

All functions modifying a list:

- Take pList as a parameter
- Return the “new” pointer to the first NODE

e.g.: `NODE *insertNode(...);`

Lists Versus Arrays

Arrays

- Elements are consecutive in memory
- Easy to determine location in memory
- Access a single element quickly
- Cannot change size (must declare new space)

Lists Versus Arrays

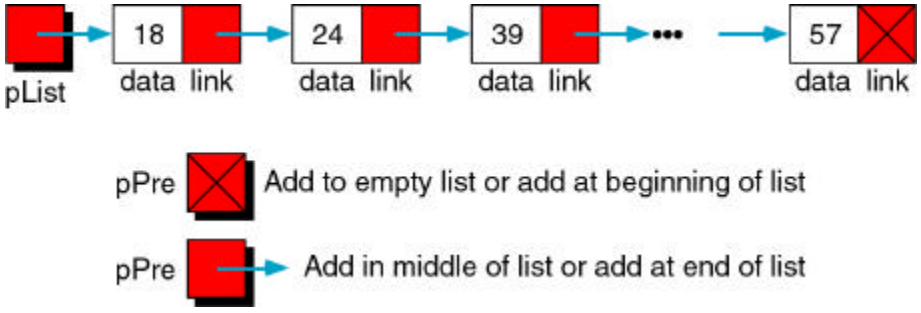
Lists

Elements are in arbitrary location
Must view all predecessors to find one
Grows to arbitrary size
Massive re-orderings possible quickly

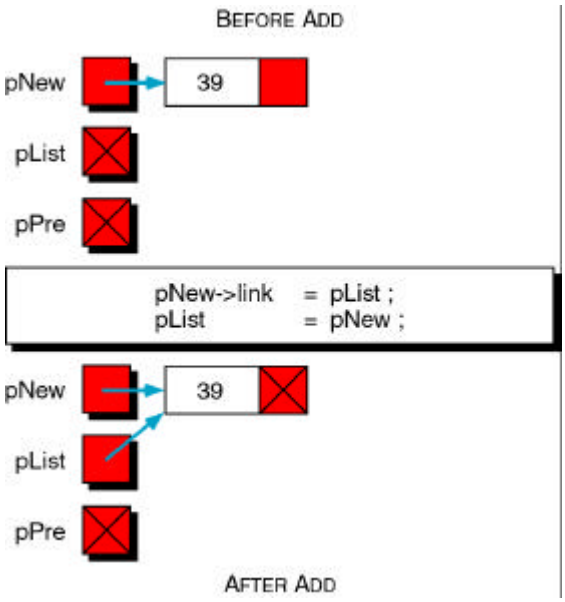
Insert a Node

- Allocate memory
- Locate predecessor (**`pPre`**)
- Point new node to its successor
- Point predecessor to new node

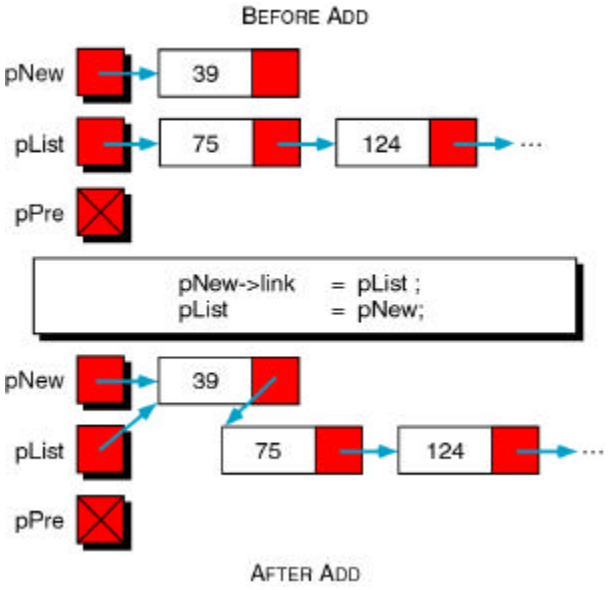
Meanings of **`pPre`**



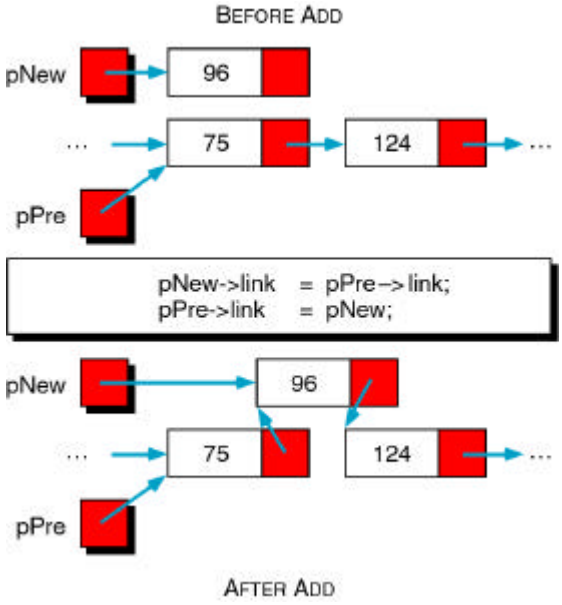
Insert into Empty List



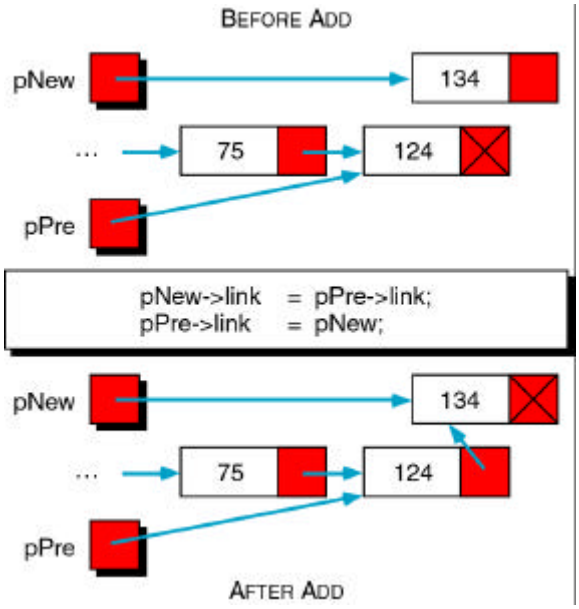
Insert Node at Beginning



Insert Node in Middle



Insert Node at End



Inserting a Node

```
NODE * insertNode(NODE *pList,
                  NODE *pPre,
                  DATA data) {
.
.
.
}
```

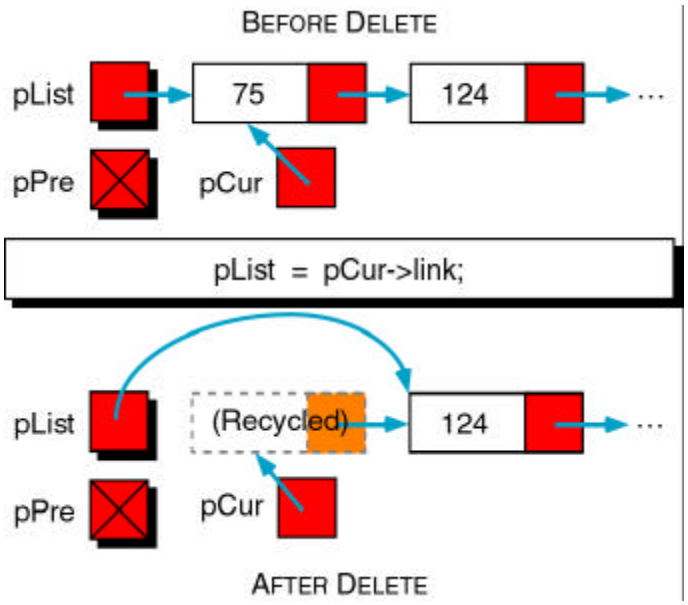
Deleting a Node

Still requires the predecessor's location!

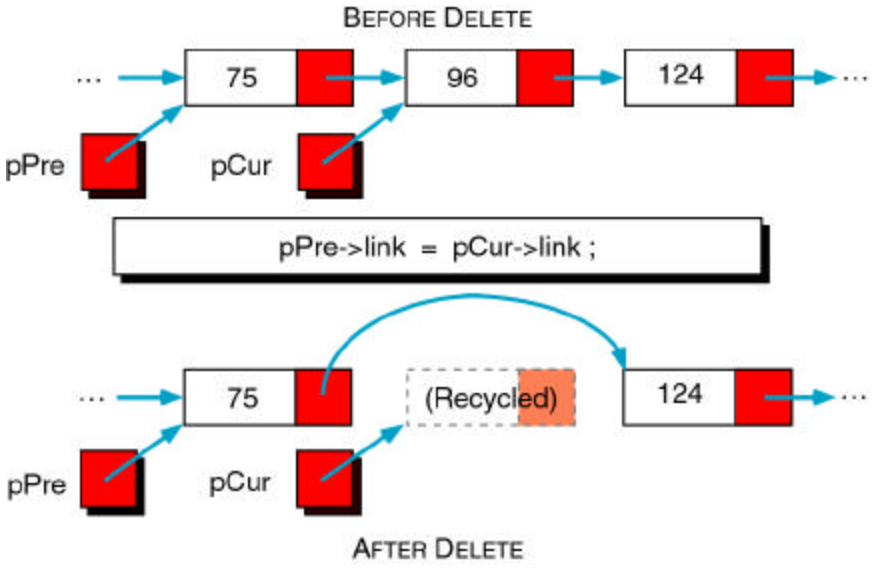
Cut out the node

Free the memory space!

Delete First Node



Delete General Case



Deleting a Node

```
NODE * deleteNode(NODE *pList,
                  NODE *pPre) {
.
.
.
}
```

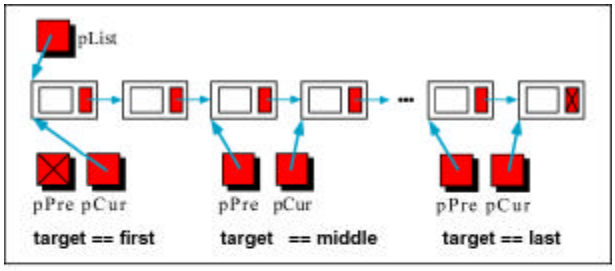
Searching Through a List

Search should return:

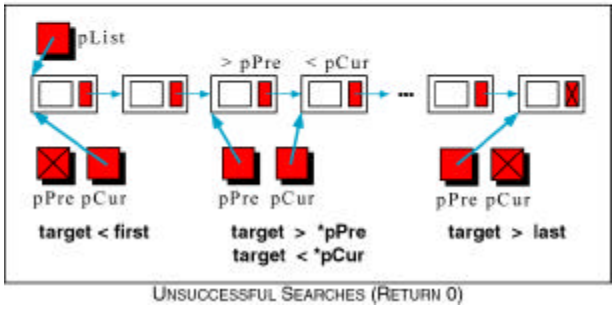
0 if no match exists

1 if **`pCur`** is a match

Search Returns:



SUCCESSFUL SEARCHES (RETURN 1)



UNSUCCESSFUL SEARCHES (RETURN 0)

searchList

```
int searchList (NODE *pList,
                NODE **pPre,
                NODE **pCur,
                KEY_TYPE target) {
.
.
.
}
```

Traverse Linked Lists

printList, averageList, …