

Self-Referential Structures

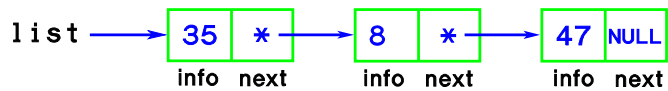
```
struct nodetype
{
    int info;
    nodetype *next;
};

nodetype *list;

nodetype *p, *q; //Used later
```

LL00010

Linked Lists



Array representation of same list:

0	35
1	8
2	47

LL00020

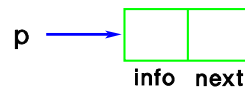
Linked List Declarations

To initialize the list:

```
list = NULL;
```

Creating a node for the list:

```
p = new nodetype;
```



LL00030

Linked List Operations

To add the first node to the list:

```
p->info = 35; // Value in node  
p->next = NULL;  
list = p;
```

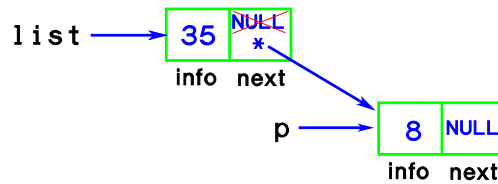


LL00040

Linked List Operations

To add the second node to the list:

```
p = new nodetype;  
p->info = 8; // Value in node  
p->next = NULL;  
list->next = p;
```

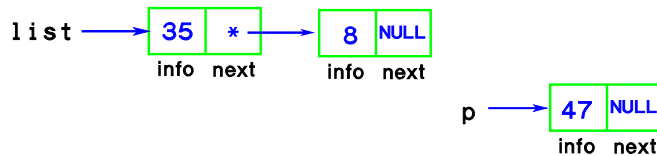


LL00050

Linked List Operations

To add a node to the end of an existing list:

```
p = new nodetype;  
p->info = 47; // Value in node  
p->next = NULL;  
q = list;  
while(q->next != NULL) } List traversal  
    q = q->next;  
q->next = p;
```



LL00060

Linked List Operations

To add a node to the end of any list (even empty):

```
p = new nodetype;
p->info = 47; // Value in node
p->next = NULL;
if(list == NULL)
    list = p;
else // traverse the list
{
    q = list;
    while(q->next != NULL)
        q = q->next;
    q->next = p;
}
```

LL00070

Linked List Operations

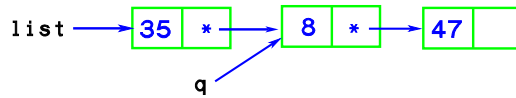
To add a node to the beginning of a list:

```
p = new nodetype;
p->info = 47; // Value in node
p->next = NULL;
if(list == NULL)
    list = p;
else
{
    p->next = list;
    list = p;
}
```

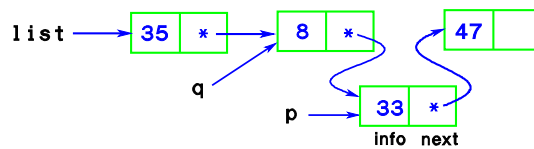
LL00080

Linked List Operations

To add the node pointed to by *p* to the list after the node pointed to by *q*:



```
p = new nodetype;
p->info = 33;
p->next = q->next;
q->next = p;
```



LL00090

Linked List Operations

To output the list (also a good example of a list traversal):

```
q = list;
while(q != NULL)
{
    cout << q->info << endl;
    q = q->next;
}
```

LL00100

Using a for Statement for List Operations

while version

```
q = list;
while(q != NULL)
{
    cout << q->info;
    q = q->next;
}
```

for version

```
for(q = list; q != NULL; q = q->next)
{
    cout << q->info;
}
```

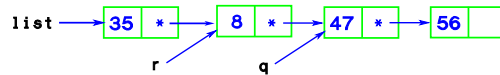
for statements are usually used only for "counting" loops. Here, the for loop conveniently contains all the necessary code to control the list traversal

LL00f05

Linked List Operations – Trailing Pointer

Since insertion operations require that the position of insertion be indicated by a pointer to the node BEFORE the insertion, a "trailing pointer" can be used. One pointer is used to find the insertion position, and another (trailing) one is used to perform the insertion:

```
q = list; r = NULL;
while(q != NULL)
{
    // Code to find the correct position goes here
    r = q; q = q -> next;
}
```

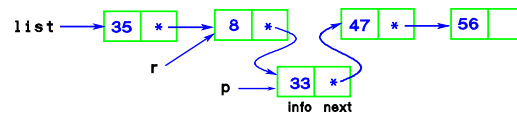


LL00f0

Linked List Operations – Trailing Pointer

To do the insertion, use the trailing pointer (*r*):

```
if (r == NULL) // insert at beginning of list
{
    p -> next = list; list = p;
}
else
{
    p -> next = r -> next; r -> next = p;
}
```



LL00120

Trailing Pointer Example

Insert a node into a list arranged in numeric order:

```
q = list; r = NULL;
while(q != NULL)
{
    if(p -> info > q -> info)
        break;
    else
    {
        r = q; q = q -> next;
    }
} // end while
if (r == NULL)
{
    p -> next = list; list = p;
}
else
{
    p -> next = r -> next; r -> next = p;
} // end if
```

LL00130

To Delete a List

```
q = list;
while(q != NULL)
{
    p = q;
    q = q -> next;
    delete p;
}
```

LL00140