Self-Referential Structures

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

nodetype *list;

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

Linked Lists

```
list → 35 * → 8 * → 47 NULL
```

Array representation of same list:

```
0 35
1 8
2 47
```
Linked List Declarations

To initialize the list:

```
list = NULL;
```

Creating a node for the list:

```
p = new nodetype;
```

<table>
<thead>
<tr>
<th>Info</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Linked List Operations

To add the first node to the list:

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

<table>
<thead>
<tr>
<th>list</th>
<th>35</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>35</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>
Linked List Operations

To add the second node to the list:

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

```
list -> 35 NULL
     ^      |
     |      |
     |      v
     v
     8 NULL
```

Linked List Operations

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

```c
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;
}
```

```
list -> 35 -> 8 NULL
       ^      |
       |      |
       |      v
       v
       47 NULL
```

```c
p
       ^      |
       |      |
       |      v
       v
```
Linked List Operations

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

```c
p = new nodelist;
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;
}
```

Linked List Operations

To add a node to the beginning of a list:

```c
p = new nodelist;
p->info = 47; // Value in node
p->next = NULL;
if(list == NULL)
    list = p;
else
{
    p->next = list;
    list = p;
}
```
**Linked List Operations**

To add the node pointed to by \( p \) to the list after the node pointed to by \( q \):

\[
\begin{align*}
\text{list} & \quad 35 & \quad 8 & \quad 47 \\
\quad & \quad q \\
p = \text{new nodetype} ; \\
p->\text{info} = 33 ; \\
p->\text{next} = q->\text{next} ; \\
q->\text{next} = p ;
\end{align*}
\]

**Linked List Operations**

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

\[
\begin{align*}
q = \text{list} ; \\
\text{while}(q != \text{NULL}) \\
\{ \\
\quad \text{cout} \ll q->\text{info} \ll \text{endl} ; \\
\quad q = q->\text{next} ;
\}
\end{align*}
\]
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.

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;
}
```

```
| 55 | 8 | 47 | 68 |
```

```
list 55 4 8 47 68 58
```

```
r q
```
Linked List Operations – Trailing Pointer

To do the insertion, use the trailing pointer $r$:

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

![Diagram of linked list operations with pointers $r$, $p$, and $q$.]

Trailing Pointer Example

Insert a node into a list arranged in numeric order:

```c
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
```
To Delete a List

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