Self-Referential Structures

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

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

Linked Lists

```
list -> 35 * -> 8 * -> 47 NULL

Array representation of same list:

<table>
<thead>
<tr>
<th></th>
<th>35</th>
<th>8</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>8</td>
<td>47</td>
</tr>
</tbody>
</table>
```
Linked List Declarations

To initialize the list:

```c
list = NULL;
```

Creating a node for the list:

```c
p = new nodetype;
```

```
   p
      ↓
  info   next
```

Linked List Operations

To add the first node to the list:

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

```
   p
      ↓
  35    NULL
      info   next
```

```
  list = p;
  list
      ↓
  35    NULL
      info   next
```
Linked List Operations

To add the second node to the list:

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

![Diagram of linked list operation](image1)

---

Linked List Operations

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

```java
p = new nodetype;
p->info = 47; // Value in node
p->next = NULL;

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

![Diagram of linked list operation](image2)
Linked List Operations

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

```c
p = new nodetype;
p->info = 47; // Value in node
p->next = NULL;
if(list == NULL)
    list = p; // adds first node to list
else  // add second, third, etc.
{
    q = list;
    while(q->next != NULL) // list traversal
        q = q->next;
    q->next = p; // link to end of list
}
```

Linked List Operations

To add a node to the beginning of a list:

```c
p = new nodetype;
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 \( q \):

```cpp
list \rightarrow \begin{array}{c}
35 \quad 8 \quad 47 \quad \cdots \\
\end{array}
```

\( \begin{array}{c}
p = \text{new nodetype} ; \\
p->\text{info} = 33 ; \\
p->\text{next} = q->\text{next} ; \\
q->\text{next} = p ; \\
\end{array} \)

```
\begin{array}{c}
35 \quad 8 \quad 47 \quad 33 \quad \cdots \\
\end{array}
```

Linked List Operations

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

```cpp
q = list ; \\
while(q != NULL) \\
{ \\
    cout << q->\text{info} << \text{endl} ; \\
    q = q->\text{next} ; \\
}
```
Using a \textbf{for} Statement for List Operations

\begin{verbatim}
while version
q = list;
while(q != NULL)
{
    cout << q->info;
    q = q->next;
}
\end{verbatim}

\texttt{for} statements are usually used only for "counting" loops. Here, the \texttt{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 \textbf{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:

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

\begin{itemize}
    \item \texttt{list} \begin{tikzpicture}[>=stealth,auto,thick]
        \node (a) at (0,0) {$35$};
        \node (b) at (1,0) {$8$};
        \node (c) at (2,0) {$47$};
        \node (d) at (3,0) {$58$};
        \node (e) at (4,0) {};\draw[<->] (a) -- (b);
        \draw[<->] (b) -- (c);
        \draw[<->] (c) -- (d);
        \draw[<->] (d) -- (e);
        \node (f) at (1.5,-1) {r};
        \node (g) at (2.5,-1) {q};
    \end{tikzpicture}\end{itemize}
Linked List Operations – Trailing Pointer

To do the insertion, use the trailing pointer method:

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

![Diagram of linked list operations](image)

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
```
To Delete a List

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