Create a List

Practice Exercise

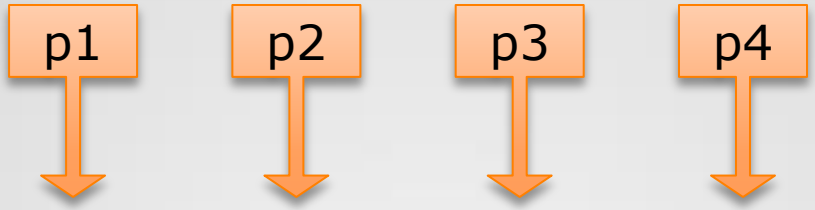
Create a List

For this practice exercise, assume that

- All the functions are members of the Node class, as in the slides.
- The overloaded constructor sets the member variable **`data`** in the node to character '**`x`**'.

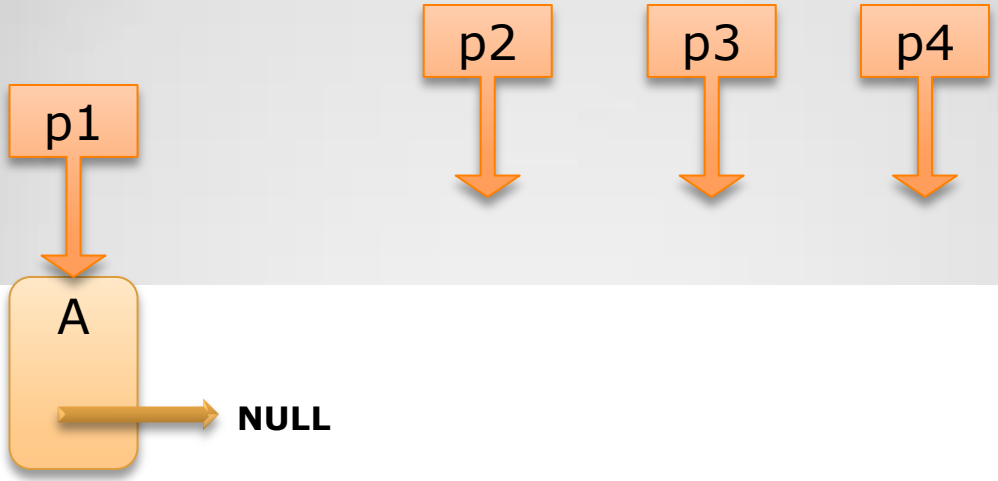
```
Node *p1, *p2, *p3, *p4;
```

```
Node *p1, *p2, *p3, *p4;    // Create four pointers that will point
                            //    to objects of the class Node.
```



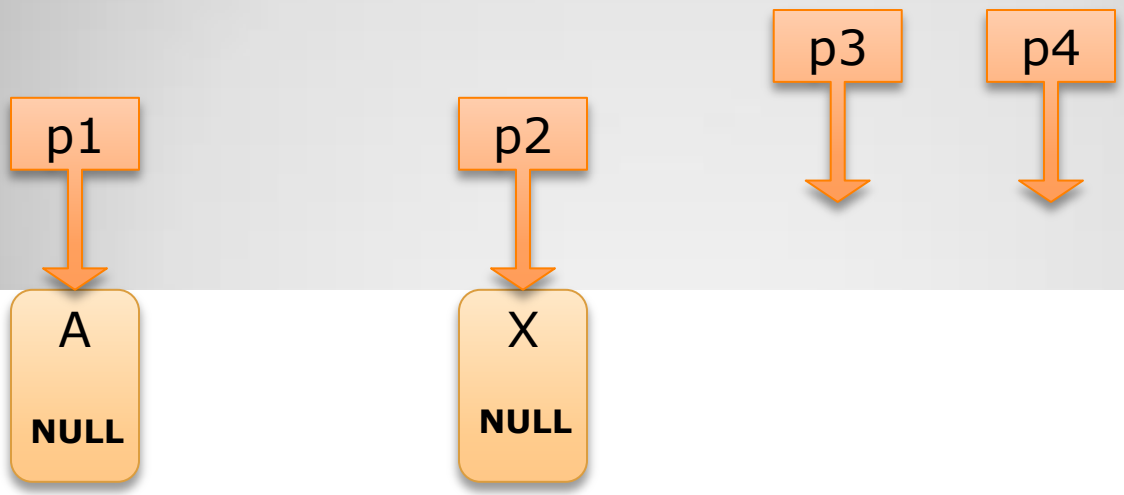
```
p1 = new Node('A', NULL);
```

```
p1 = new Node('A', NULL);   // Using the overloaded constructor,
                            //    create a complete new node and
                            //    set p1 to point to it.
```



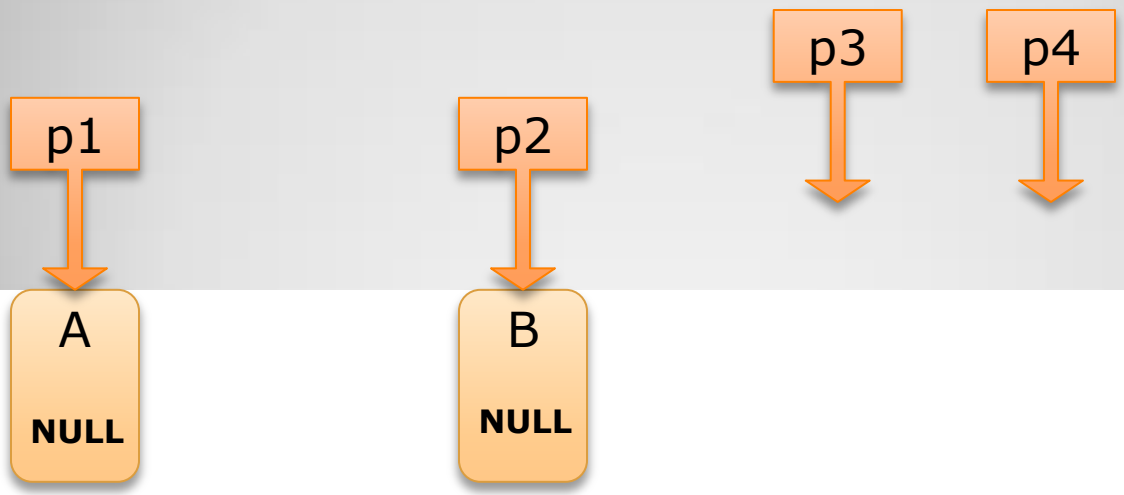
```
p2 = new Node;
```

```
p2 = new Node;      // Create another new node and set p2 to point to it.
                    // The statement calls the default constructor of the
                    //     class Node, which sets the data to 'X' and the
                    //     pointer to NULL.
```



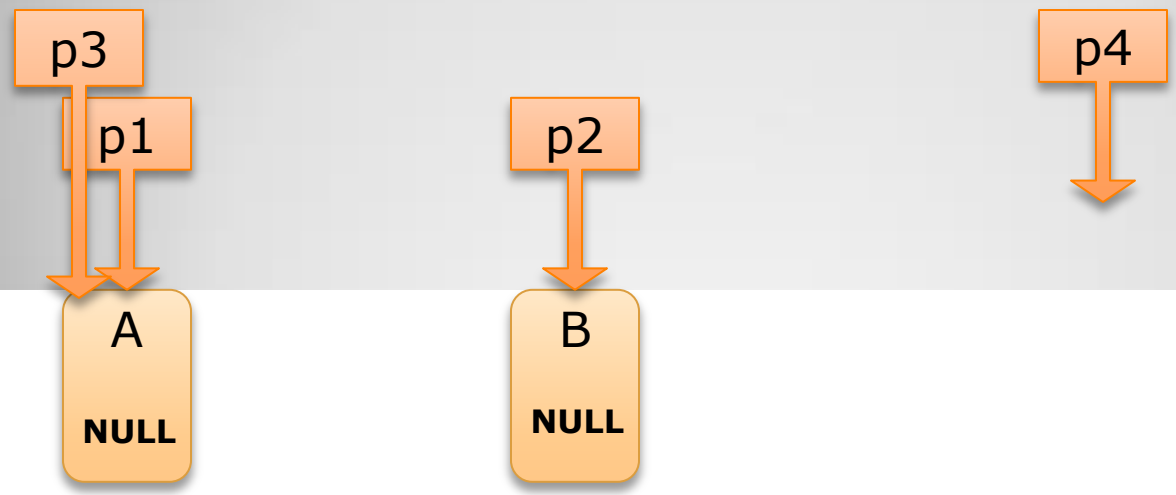
```
p2->setData('B');
```

```
p2->setData('B'); // Stores char 'B' in the node that p2 is pointing to.
```



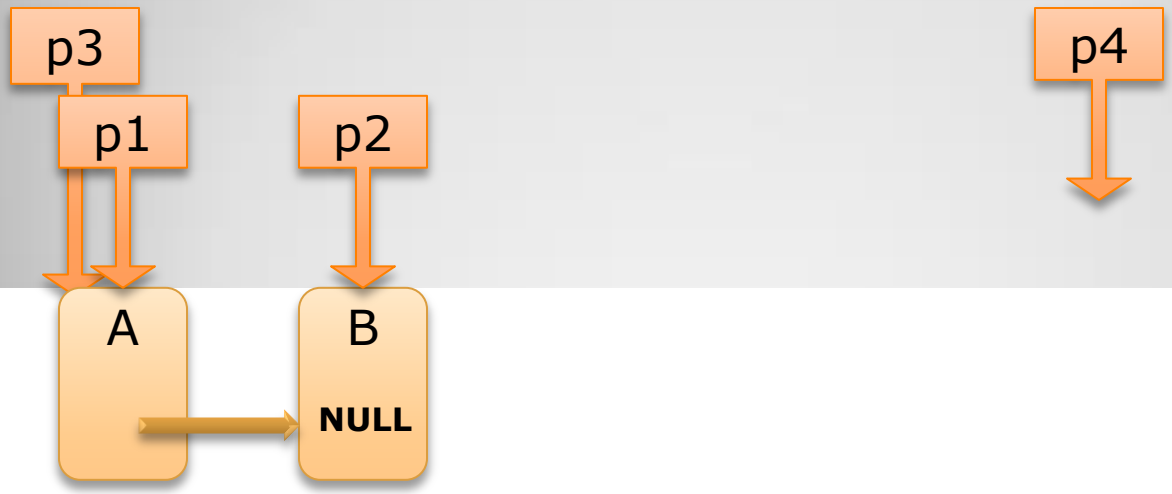
```
p3 = p1;
```

```
p3 = p1; // Stores in pointer p3 the same address that it is stored in
         //    pointer p1. Pointer p3 will now point to the same node
         //    p1 is pointing to.
```



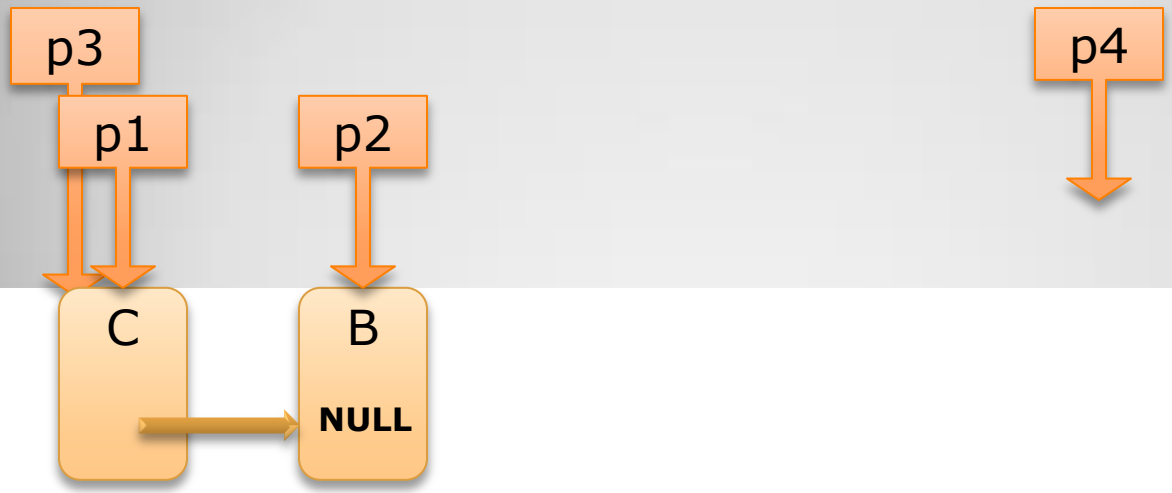
```
p1->setPtrToNext(p2);
```

```
p1->setPtrToNext(p2);  // Sets the pointer in the node pointed by p1
                       //     to point to the node pointed by p2.
```



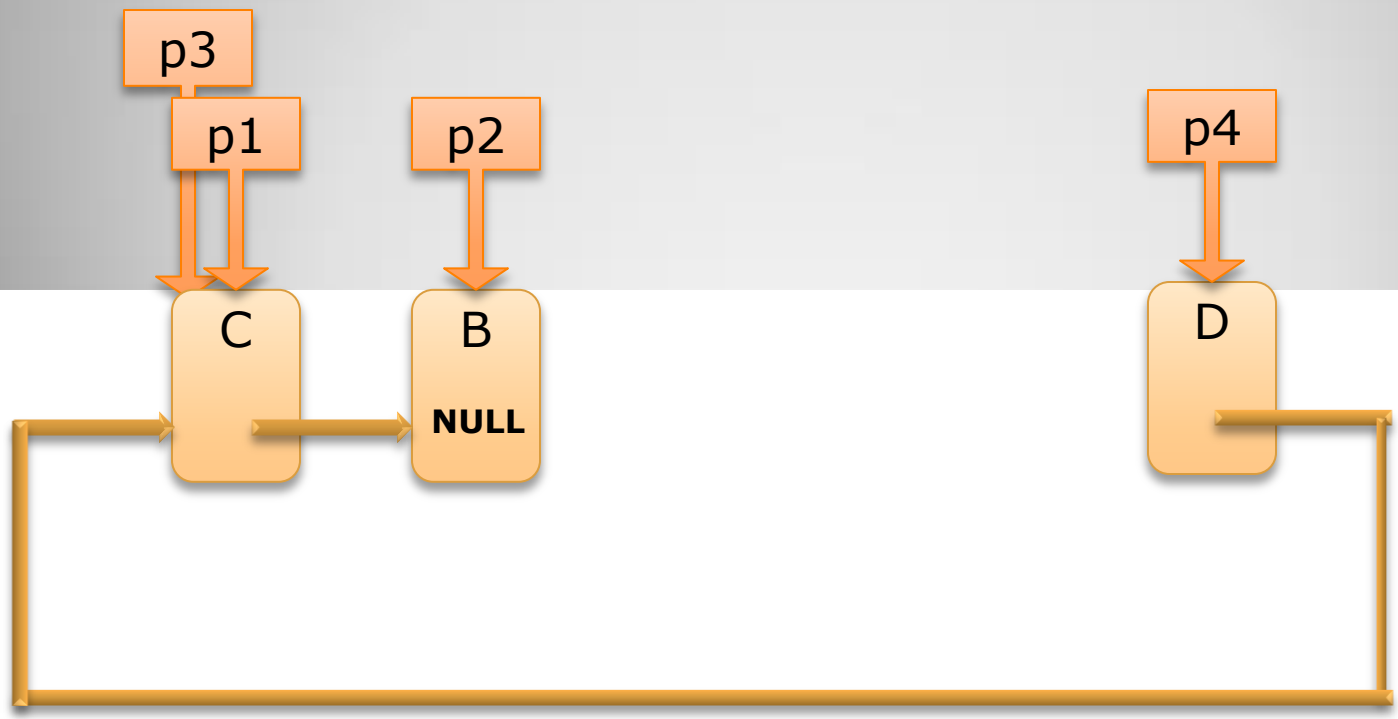
```
p3->setData('C');
```

```
p3->setData('C'); // Overwrites the data store in the node pointed
                  //    by p3. The same result can be obtained if we
                  //    used the statement p1->setData('C');
```



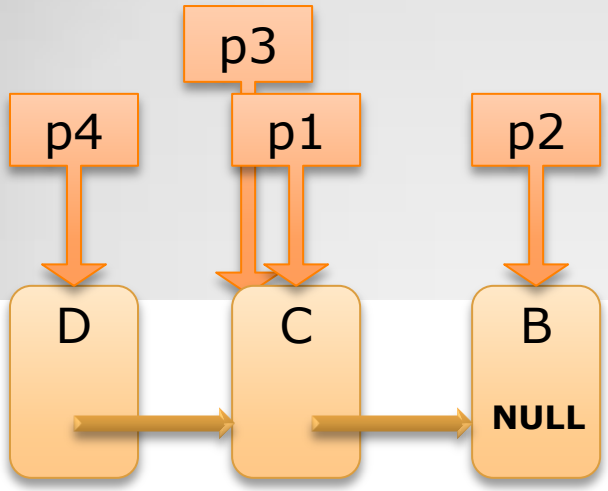
```
p4 = new Node('D', p1);
```

```
p4 = new Node('D', p1); // Creates a new node. Pointer p4 will point
                        //    to the new node. The node stores the
                        //    character 'D' and points to the node
                        //    that p1 is pointing to.
```



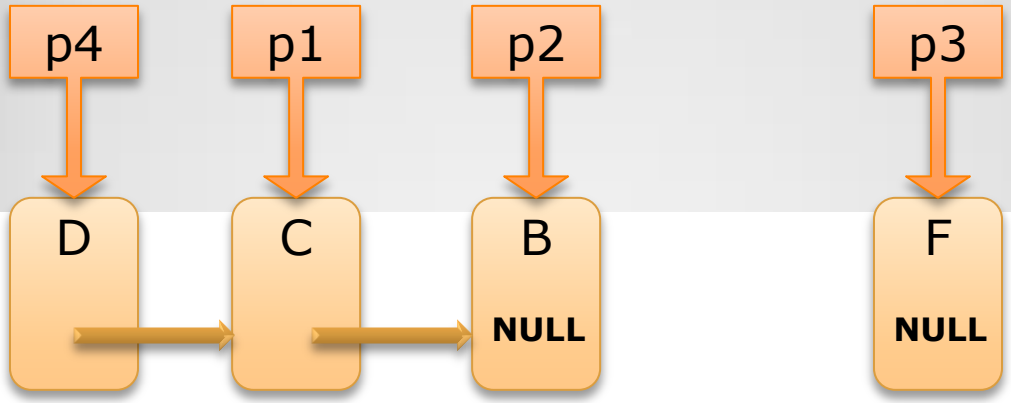
```
// Re-arrange.
```

// Re-arrange.



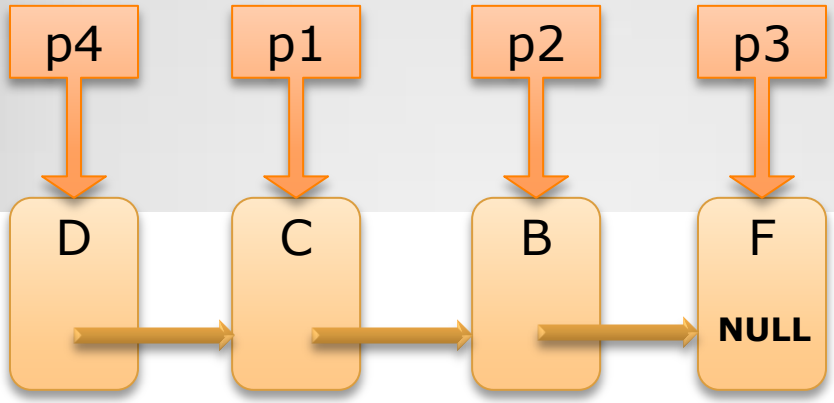
```
p3 = new Node('F', NULL);
```

```
p3 = new Node('F', NULL); //Creates a new node. Pointer p3 will
                          //   point to the new node. The node
                          //   stores the character 'F' and the
                          //   pointer is set to NULL.
```



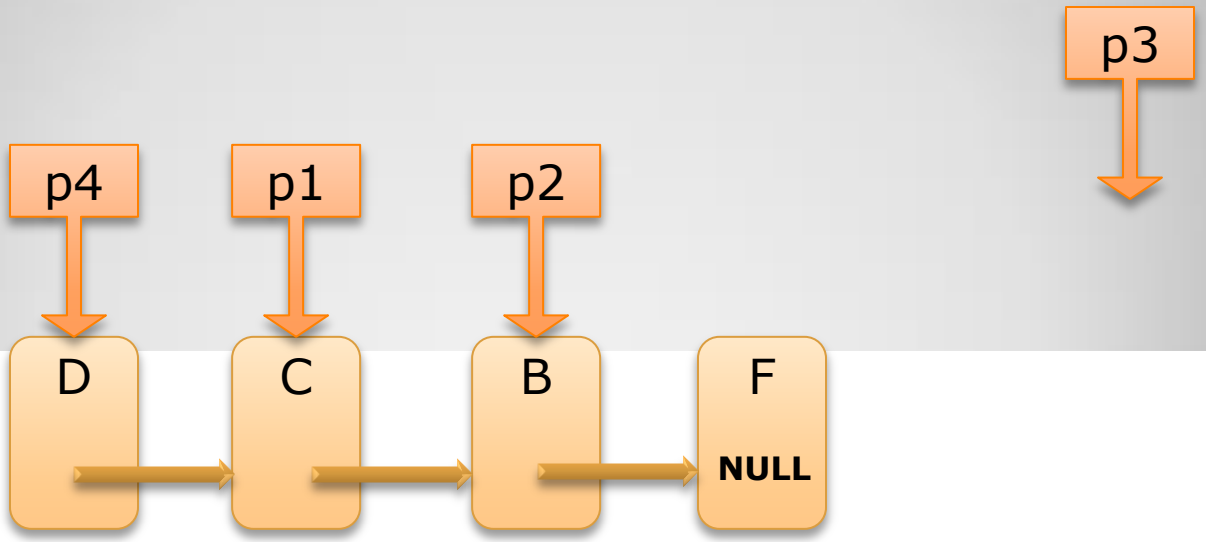
```
p2->setPtrToNext(p3);
```

```
p2->setPtrToNext(p3);   // Sets the pointer in the node pointed by p2
                        //    to point to the node pointed by p3.
```



```
p3 = NULL;
```

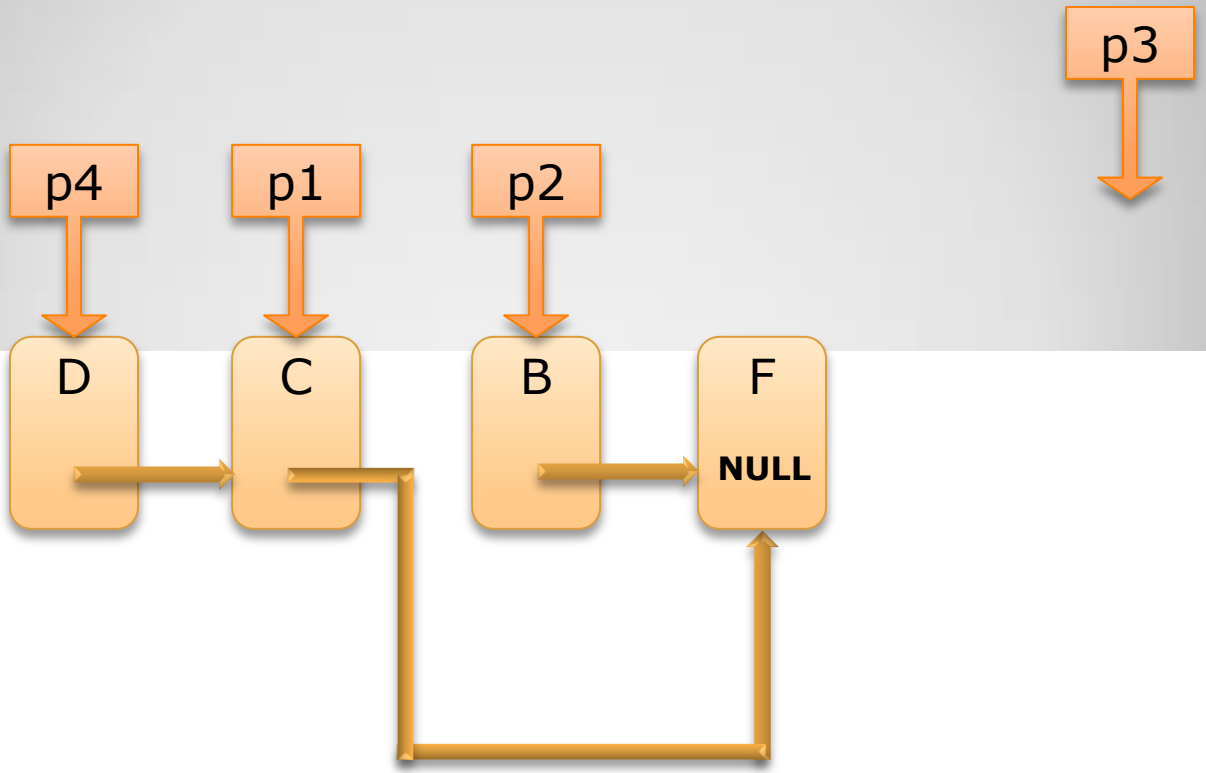
```
p3 = NULL; // Pointer p3 does not point to any node anymore.
```



```
p1->setPtrToNext(p2->getPtrToNext());
```

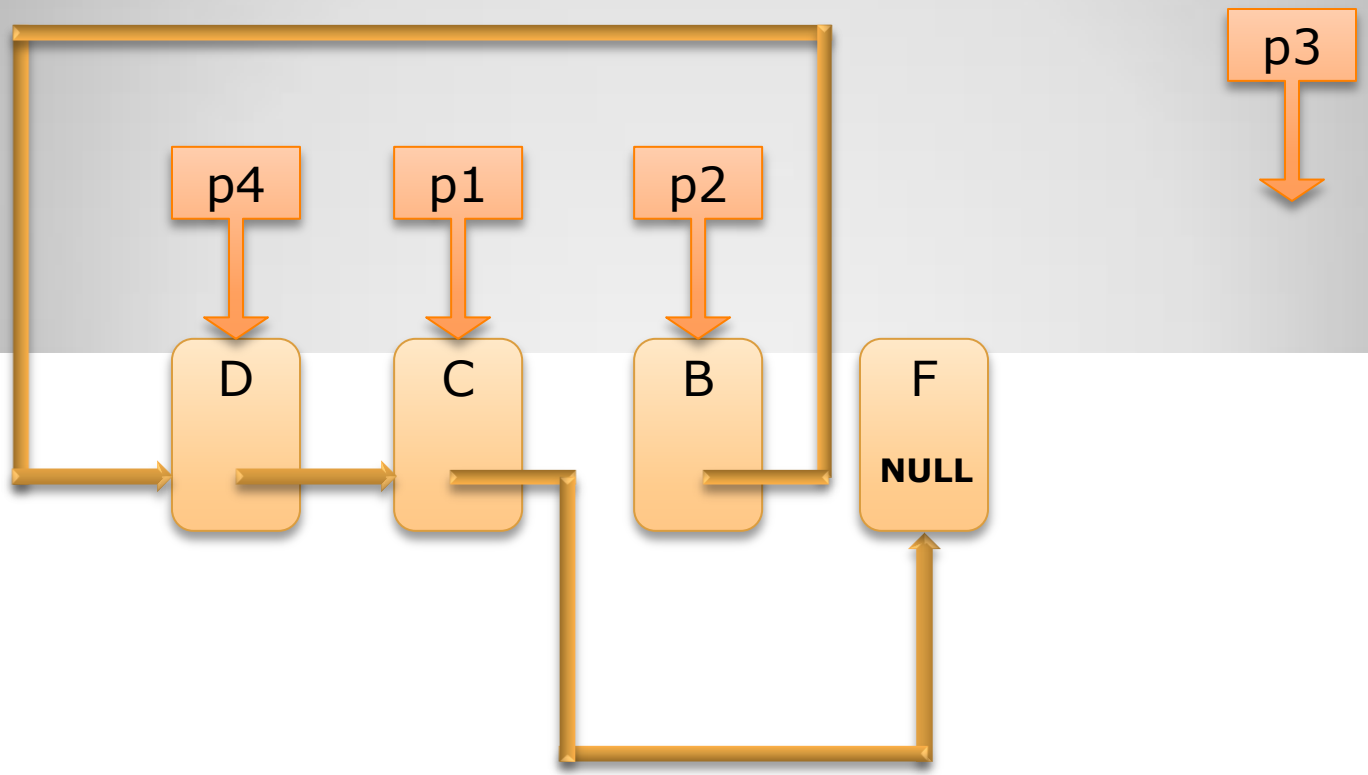
```
p1->setPtrToNext(p2->getPtrToNext());
```

```
// Sets the pointer in the node pointed by p1
//    to point to the node pointed by the node
//    that is pointed by p2.
```



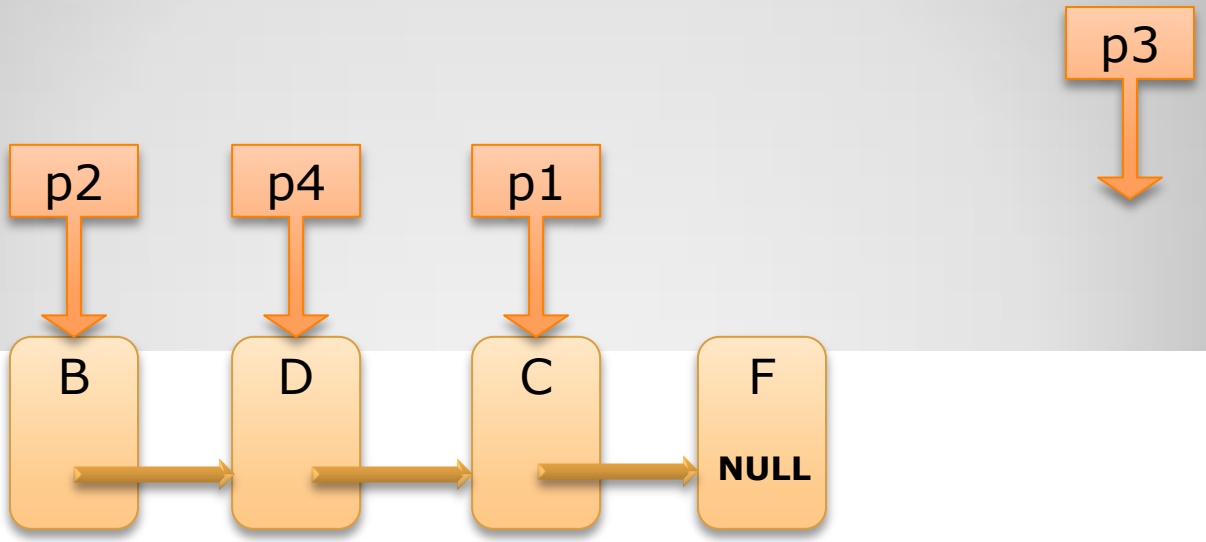
```
p2->setPtrToNext(p4);
```

```
p2->setPtrToNext(p4);  // Sets the pointer in the node pointed by p2
                       //     to point to the node pointed by p4.
```



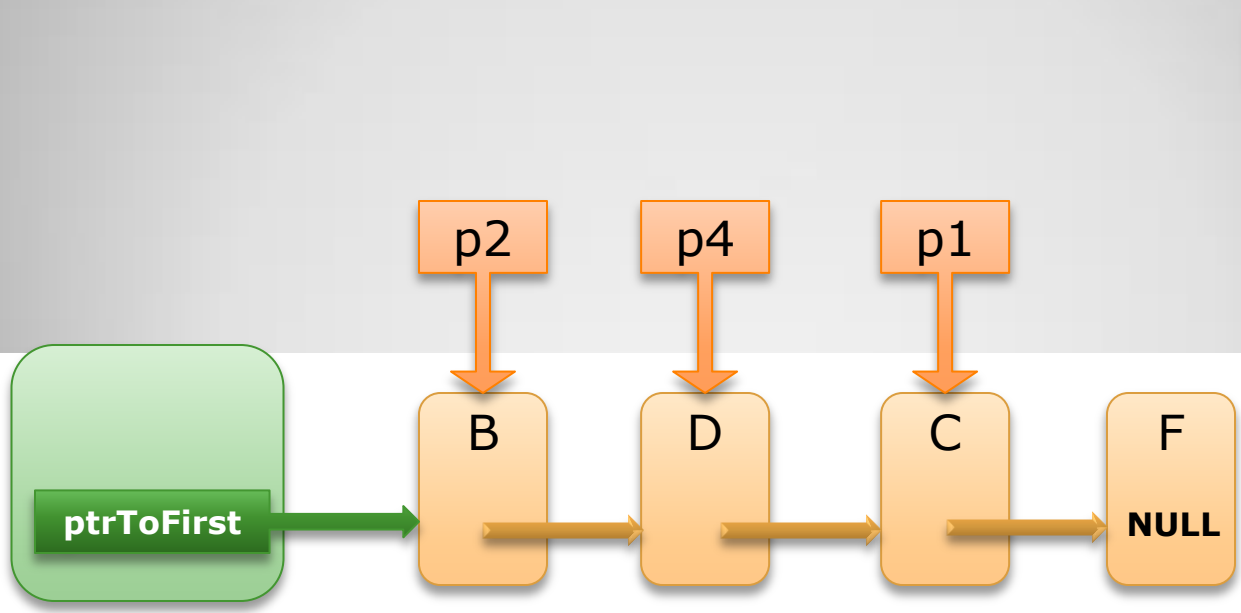
```
//Re-arrange.
```

//Re-arrange.



```
prtToFirst = p2
```

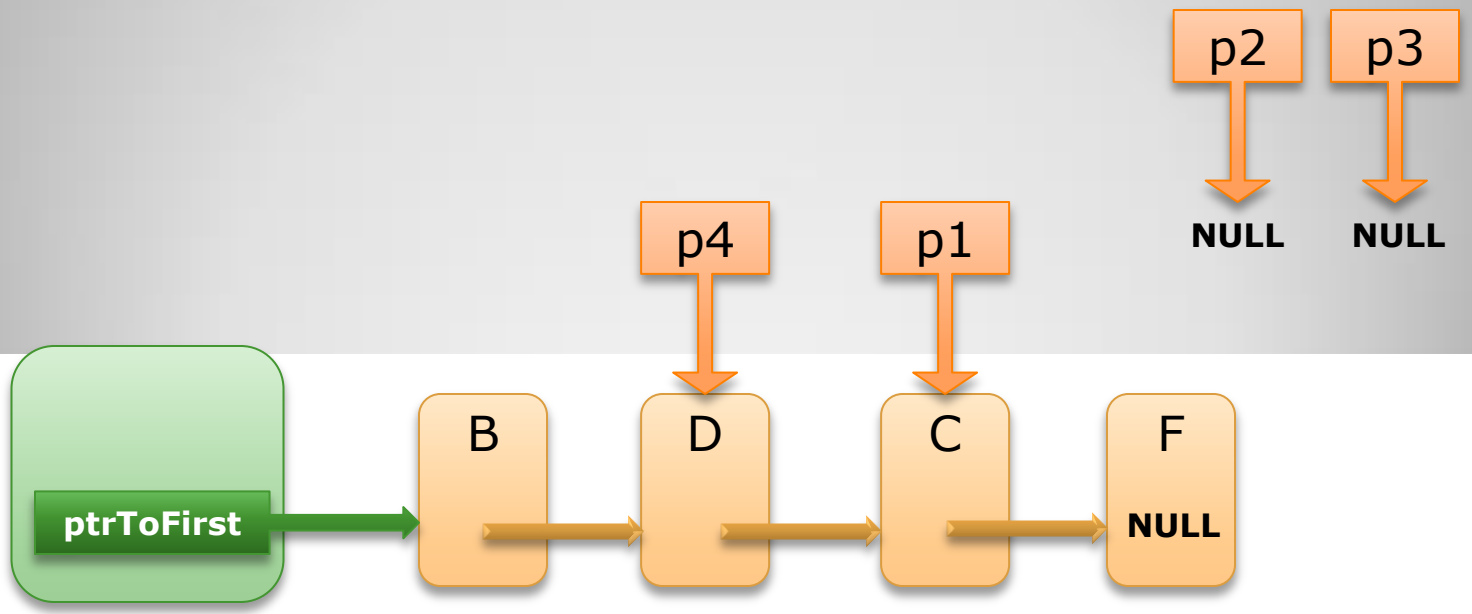
```
prtToFirst = p2   // The list object will point to the node that p2 is
                  //    pointing to, and this node will become the
                  //    first of the list.
```



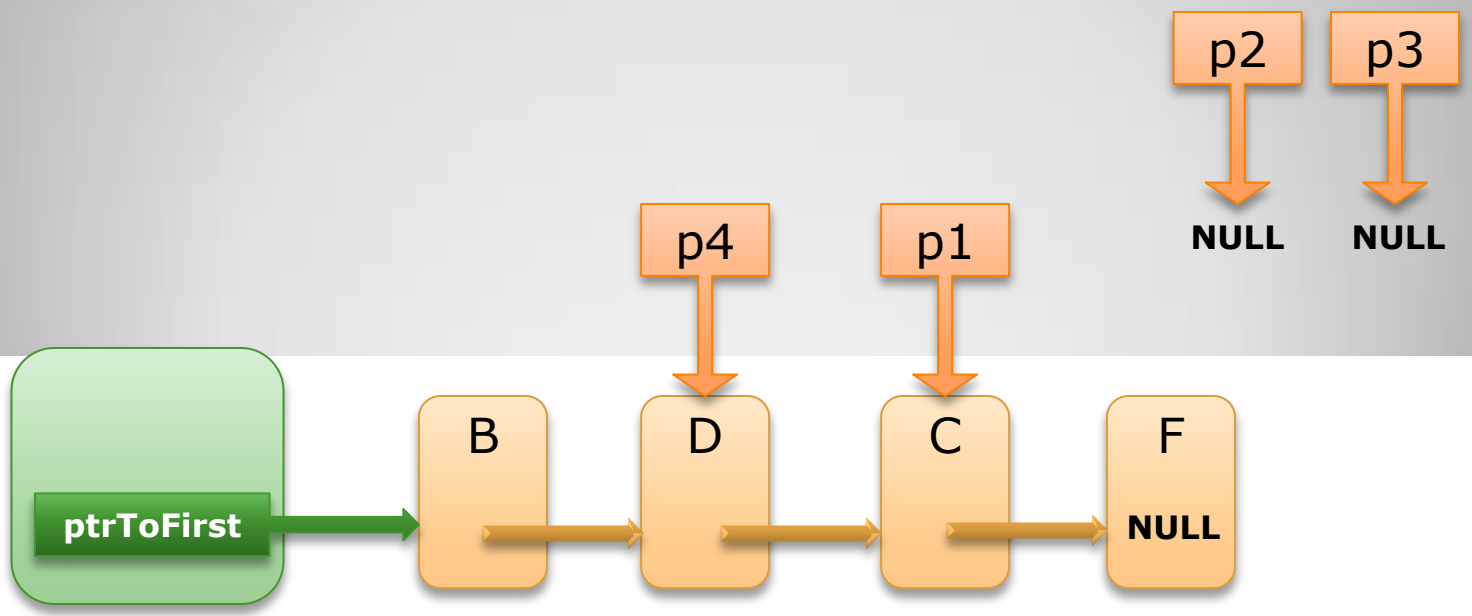

```
// The list object should also contain a variable to store the number
//    of nodes, but that was not part of this exercise.
```

```
p2 = NULL;
```

```
p2 = NULL;   // Pointer p2 does not point to any node anymore.
```



```
count = 4;
```



This is the final list.