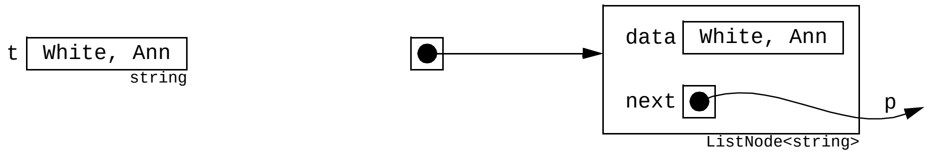
The constructor creates a new node, assigning the `T` value `t` to its `data` field and the pointer `p` to its `next` field:

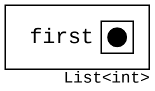


If `T` is a class (instead of an ordinary type), its constructor will be called by the declaration of `data`. Note that the class `List<T>` is declared here to be a `friend` of the `ListNode` class. This will allow the member functions of the `List` class to access the protected members of the `Node` class.

Here is the `List` class template interface:

```
template<class T>
class List
{ public:
    List() : first(0) { }
    ~List();
    void insert(T t);      // insert t at front of list
    int remove(T& t);      // remove first item t in list
    bool isEmpty() { return (first == 0); }
    void print();
  protected:
    ListNode<T>* first;
    ListNode<T>* newNode(T& t, ListNode<T>* p)
    { ListNode<T>* q = new ListNode<T>(t,p); return q; }
};
```

A `List` object contains only the pointer `first`:



This points to a `ListNode` object. The default constructor initializes the pointer to `NULL`. After items have been inserted into the list, the `first` pointer will point to the first item in the list.

The `newNode` function invokes the `new` operator to obtain a new `ListNode` object by means of the `ListNode()` constructor. The new node will contain the `T` value `t` in its `data` field and the pointer `p` in its `next` field. The function returns a pointer to the new node. It is declared `protected` because it is a utility function that is used only by the other member functions.

The `List` destructor is responsible for deleting all the items in the list:

```
template<class T>
List<T>::~List()
{ ListNode<T>* temp;
  for (ListNode<T>* p = first; p; )          // traverses list
  { temp = p;
    p = p->next;
    delete temp;
  }
}
```