## The Node class

In C++, the Node class serves as a fundamental building block for linked data structures such as linked lists. Here's a brief overview of  its functionality and purpose:

| | |
|---|---|
| **Purpose:**<br>`class Node`<br>`{`<br><br>`};` | The class is designed to store individual elements of a **linked list**, each containing data and a pointer to the next node, thus forming a chain.<br><br>The **Node** class is typically used in situations where you need a dynamically constructed list, each element of which can independently contain data and point to the next element in the sequence. We will use this Node class in implementing the **CadidateList** Class. |
| **Default Constructor**<br><br>**Node()** | The default constructor initializes the node with its link member set to **nullptr,** indicating that it does not yet point to any next node. This is typical for the last node in **a linked list** or when a node has not yet been linked. |
| **Overloaded Constructor**<br><br>**Node(const CandidateType& votes, Node* theLink)** | This constructor takes two arguments: **votes** and **theLink**. **votes** is an object of type **CandidateType** (holding data related to a candidate, such as votes). **theLink** is a pointer to the next **Node** in the list. This constructor initializes a node with provided data and link, allowing for **dynamic insertion** into the list. |
| **getLink()** | Returns the pointer to the next node. This function is useful for **traversing the list** from one node to another. |
| **getCandidate():** | Returns the data (of type **CandidateType**) stored in the node. This is useful when you need to access or process the specific data held by the node. |
| **setCandidate(const CandidateType& votes)** | Allows updating the data in the node. This can be used to modify the node's data without altering the structure of the list. |
| **setLink(Node theLink)** | Sets the node's link to another node. This function is **critical for re-linking nodes**, either when inserting new nodes into the list or rearranging existing parts of the list. |
| **Destructor** | The destructor here is trivial as it does nothing special—it is empty, suggesting that no special cleanup is needed when a **Node** is destroyed. |