Deleting a Node From a List

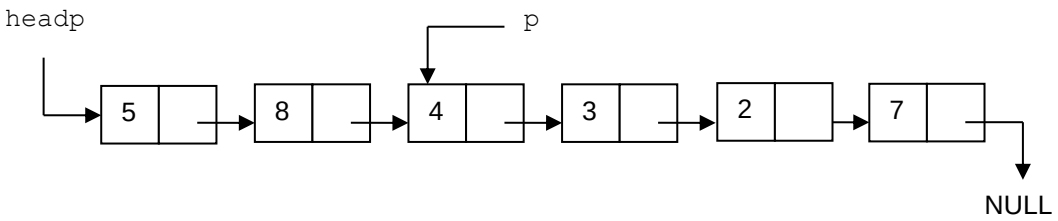
- The steps of deleting a node from a list are as follows:

  1. Remember the address of the node you want to delete
  2. Store the data in the node you want to delete
  3. Form the new links
  4. Delete the node from the memory (using `free` function)

- Similar to add operation, we will examine the delete operation in four cases:

  1. Deleting a node between two nodes
  2. Deleting the last node of a list
  3. Deleting the first node of a list
  4. Deleting the only node of a list with a single node

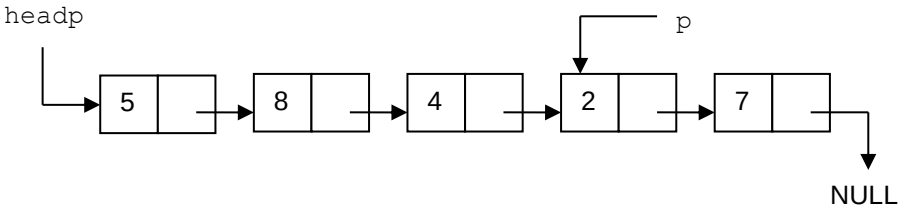
- In order to delete a node between two nodes, we need to know the address of the node which comes before the node we want to delete. For example, let's delete the node after the node pointed by `p` in the list below. Thus, we want to delete 3.



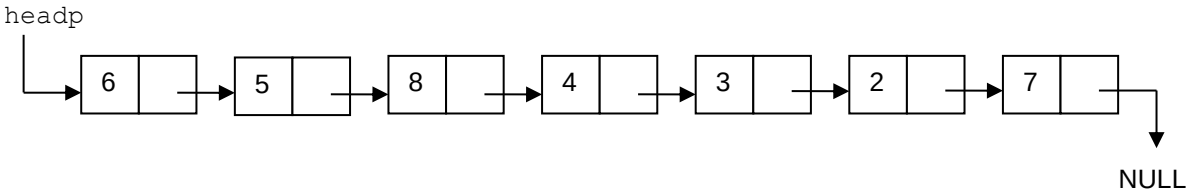
- We need to remember the address of the node we want to delete, so that we can free its memory location after the delete operation. Therefore, we need an additional variable to store that address.

```
void deleteAfter(node_t *p, Ltype *item)
{
      node_t *del;
      /* remember the address of the node you want to delete,
         thus the node after p */
      del = p->next;
      /* store the data in the node you want to delete in
         *item */
      *item = del->data;
      /* form the new links. link the node before the one to
         be deleted to the node that comes after it. */
      p->next = del->next;
      /* delete the node from the memory */
      free(del);
}
```

- Can we use the same function to delete the last node of a list, for instance, if we want to delete 7 from the list below?



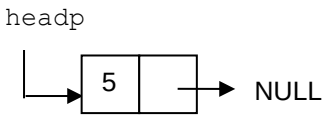
- Can we use the `deleteAfter` function to delete the first node of a list, for example to delete 6 from the the list below?



- No, because the function needs `p`, the address of the node before the node to be deleted, but there is no node before the first one. If we send `headp` to the function, it will delete 5, not 6. Therefore, similar to add operation, we need to define a separate function to delete the first node of a list. This function needs the address of the first node, thus `headp`, as a parameter.

```
node_t *deleteFirst(node_t *headp, Ltype *item) {
      node_t *del;
      /* Remember the address of the first node */
      del = headp;
      /* Store the data in the first node in *item */
      *item = del->data;
      /* Form the new links. HEAD must point to the second
         node. */
      headp = del->next;
      /* Delete the first node from the memory */
      free(del);
      return (headp);
}
```

- Can we use the `deleteFirst` function to delete the only node of a list with a single node, so that the list will become empty? For example, can we use it to delete 5 from the the list below?



- Therefore, when we are dealing with forward linked lists, we need to define two different delete functions: `deleteFirst`, which deletes the only node of a list with a single node, or the first node of any list, and `deleteAfter`, which deletes a node between two nodes, or the last node of any list.

Example: Define a function that adds a new item to the end of a list.

```
node_t *addEnd(node_t *headp, int item)
{
      node_t *p;
      /* If the list is empty, add the item at the beginning
         of the list */
      if (headp == NULL)
           headp = addBeginning(headp, item);
      else
      {    /* find the address of the last node */
           /* start from the beginning */
           p = headp;
           /* repeat until a node that points to NULL
              (i.e., the last node) is found */
           while (p->next != NULL)
                /* pass to the next node */
                p = p->next;
           /* add the item after that node */
           addAfter(p, item);
      }
      return (headp);
}
```

- Notice that, if we did not make the if check, it would also cause a problem in the condition of the while loop, because in an empty list `headp->next` is undefined.

Example: Define a function that deletes the last node of a list.

```
node_t *deleteLast(node_t *headp, int *item)
{
      node_t *p;
      /* Check if the list is not empty */
      if (headp != NULL)
           /* If there is only one node in the list, the last
              node is also the first node */
           if (headp->next == NULL)
                headp = deleteFirst(headp, item);
           else
           {    /* find the address of the node before the last
                   node */
                p = headp;
                while (p->next->next != NULL)
                    p = p->next;
                /* delete the node after it, thus last node */
                deleteAfter(p, item);
           }
      else // if the list is empty, put a dummy value in item
           *item = -987654321;
      return (headp);
}
```

- Notice that, if we did not make the second if check, it would also cause a problem in the condition of the while loop, because in a list with a single node `headp->next->next` is undefined.

Example: Define a function that deletes the n^{th} node of a list.

- In order to delete the n^{th} node we must know the address of the $(n-1)^{st}$ node, so that we can use `delete_after` function. But, if we are required to delete the first node, thus if n is 1, then we need to use `delete_first` function. If the list is empty, or if there are not that many nodes in the list, nothing can be deleted, you should return a dummy value.

```
node_t *deleteNth(node_t *headp, int n, int *item)
{
      node_t  *p;
      int k;
      if (headp != NULL)
           if (n == 1)
                headp = delete_first(headp, item);
           else
           {    /* find the address of the (n-1)st node */
                p = headp;
                k = 1;
                /* Repeat until the last node or the (n - 1)st
                   node is reached */
                while (p->next != NULL && k < n - 1)
                {
                   p = p->next;
                   k++;
                }
                /* If the last node is reached before the
                   (n - 1)st node, there are less than n nodes
                   in the list; return a dummy value */
                if (p->next == NULL)
                   *item = -987654321;
                else
                   /* delete the nth node */
                   delete_after(p, item);
           }
      else
           *item = -987654321;
      return (headp);
}
```

Example: Define a function to destroy a list.

```
node_t *destroyList(node_t *headp)
{
      node_t   *p;
      while (headp != NULL)
      {
           p = headp;
           headp = headp->next;
           free(p);
      }
      return (headp);
}
```