Examples with Linked Lists

Example: Define a function that finds the sum of all items in a given integer linked list.

```
int sumList(node_t *headp)
{
      node_t *p;
      int sum = 0;
      p = headp;
      while (p != NULL){
           sum += p->data; /* add the data in the current node */
           p = p->next;
      }
      return (sum);
}
```

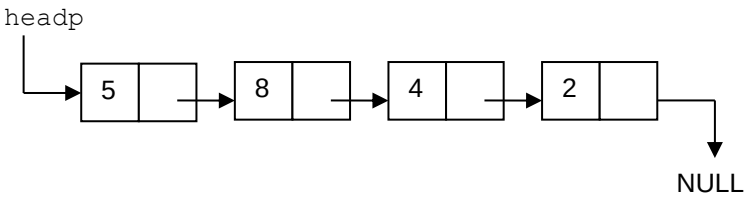
Example: Define a recursive function that finds the sum of all items in a given integer linked list.

```
int recSumList(node_t *headp)
{
      int sum;
      /* If the list is empty, the result is zero */
      if (headp == NULL)
           sum = 0;
      /* otherwise, the result is the first item plus the sum
         of the items in the list following the first node */
      else
           sum = headp->data + recSumList(headp->next);
      return (sum); /* Return the result */
}
```

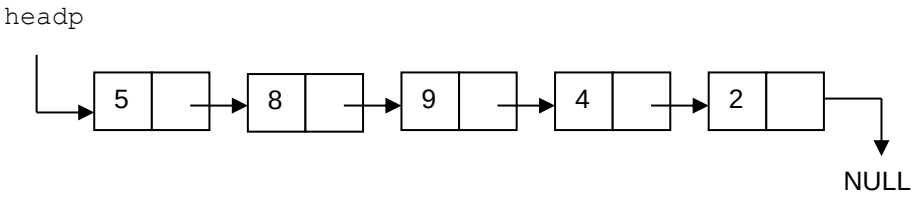
Example: Concatenate two linked lists.

```
node_t *concatLists(node_t *head1, node_t *head2)
{
      node_t  *p;
      /* if the first list is empty */
      if (head1 == NULL)
           return (head2);
      /* if the second list is empty */
      else if (head2 == NULL)
           return (head1);
      else{     /* find the last node of the first list */
           p = head1;
           while (p->next != NULL)
                p = p->next;
           /* the last node of the first list will point to
              the second list */
           p->next = head2;
           return (head1);
      }
}
```

Example: Given a linked list add a new node with value `item2`, after the node containing `item1`. For instance, if `item1` is 8, `item2` is 9, and the given list is:



- After the operation the list should look like as follows:



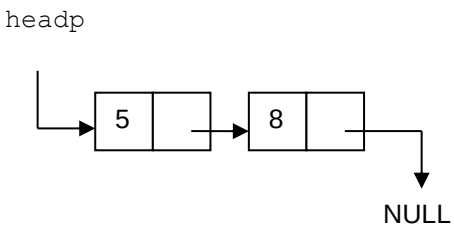
- We need to check the value returned from the `searchNode` function, before calling the `addAfter` function:

```
void add(node_t *headp, int item1, int item2)
{
      node_t *p;
      // find the address of the node containing item1
      p = searchNode(headp, item1);
      // if it is found, add the new node with item2 after it
      if (p != NULL)
           addAfter(p, item2);
}
```

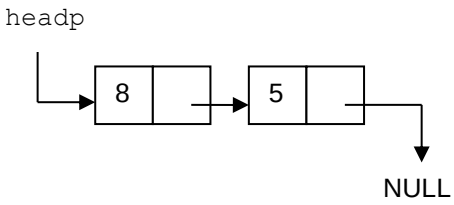
Example: Define a function that searches for an item in a sorted list.

```
node_t *searchSorted(node_t *headp, int item)
{
      node_t *p;
      /* start from the beginning of the list */
      p = headp;
      /* repeat until the end of the list is reached or a
         value which is greater than or equal to the item is
         found */
      while (p != NULL && p->data < item)
           p = p->next; /* pass to the next node */
      /* If the end of the list is reached or a value which
         is greater than the item is found, item does not
         exist in the list; return NULL. */
      if (p == NULL || p->data > item)
           return (NULL);
      else
           /* If a value which is equal to the item is found,
              return the address of that node */
           return (p);
}
```

Example: Given a linked list with two nodes, exchange their places. For instance, if the given list is



- After the operation the list should look like as follows:



- This can be done by swapping the pointers of the two nodes. As you know, for swapping operations we need to use a temporary variable.

```
node_t *swap(node_t *headp)
{
      node_t *temp;
      /* let temp point to the second node */
      temp = headp->next;
      /* let the first node point to NULL */
      headp->next = NULL;
      /* let the second node point to the first node */
      temp->next = headp;
      /* let head point to the second node */
      headp = temp;
      return (headp);
}
```

- Alternative Solution:

```
node_t *swap(node_t *headp)
{
      node_t *temp;
      /* let temp point to the first node */
      temp = headp;
      /* let head point to the second node */
      headp = headp->next;
      /* let the first node point to NULL */
      temp->next = NULL;
      /* let the second node point to the first node */
      headp->next = temp;
      return (headp);
}
```

- In the above solutions, we exchanged the places of the nodes physically. We could solve the problem by exchanging only the data of the nodes, as follows:

```
void swap(node_t *headp)
{
      int temp;
      temp = headp->data;
      headp->data = headp->next->data;
      headp->next->data = temp;
}
```

Home Exercise: Reverse a linked list.