```
// ***
// *** You MUST modify this file
// ***

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "list.h"

#ifdef TEST_READLIST
// read line by line from the input file
// each line shorter than WORDLENGTH (including '\n' and '\0')
// arithlist should have memory to store head and tail
// If arithlist is NULL, return false
// If fopen fails, return false
// If a line is too long,
//    free memory of the list
//    fclose
//    return false
// If everything is fine
//    fclose
//    arithlist points to the head and tail of the list
//    return true
bool readList(char * filename, List * arithlist)
{
  char temp[WORDLENGTH];
  FILE * fptr = fopen(filename, "r");

  if(fptr == NULL)
  {
   fclose(fptr);
   return false;
  }

  while(fgets(temp, WORDLENGTH, fptr) != NULL)
  {
   if(strchr(temp, '\n') == NULL)
    {
     free(arithlist);
     fclose(fptr);
     return false;
    }
   else
    {
     addNode(arithlist, temp);
    }
  }

  fclose(fptr);
  return true;
```

```
}
#endif

#ifdef TEST_DELETELIST
// If arithlist is NULL, do nothing
// release the memory of every node in the list
// release the memory of the list
void deleteList(List * arithlist)
{
 ListNode * pntr = arithlist -> head;

 while(pntr != NULL)
 {
  ListNode * p = pntr -> next;
  free(pntr);
  pntr = p;
 }

 free(arithlist);
}
#endif

#ifdef TEST_ADDNODE
// Input:
// arithlist stores the addresses of head and tail
// If arithlist is NULL, do nothing
// word is the word to be added
//
// Output:
// a ListNode is added to the end (become tail)
//
// allocate memory for a new ListNode
// copy word to the word attribute of the new ListNode
// insert the ListNode to the list
void addNode(List * arithlist, char * word)
{
 //List * head = arithlist;

 //insert at the tail
 if(arithlist -> head == NULL && arithlist -> tail == NULL)
 {
  ListNode * new = malloc(sizeof(ListNode));
  strcpy(new -> word, word);
  arithlist -> head = new;
  arithlist -> tail = new;
  new -> next = NULL;
  new -> prev = NULL;
 }

 else
```

```
 {
  ListNode * new = malloc(sizeof(ListNode));
  strcpy(new -> word, word); //copies word into word value of new node
  new -> prev = arithlist -> tail; //sets previous of new node to
pointer
  new -> next = NULL; //sets next of new node to null
  arithlist -> tail -> next = new; //sets next at pointer to new node
  arithlist -> tail = new; //sets tail of whole list to the new node
 }

}
#endif

#ifdef TEST_DELETENODE
//  Input:
// arithlist stores the addresses of head and tail
// If arithlist is NULL, return false
// If the list is empty (head and tail are NULL), return false
// ln is the node to be deleted
// If ln is not in the list, return false
//
// Output:
// arithlist stores the addresses of head and tail
//   after ln is deleted
// return true.
//
// Be careful about delete the first or the last node
bool deleteNode(List * arithlist, ListNode * ln)
{
  if(arithlist == NULL)
  {
   return false;
  }

  if(arithlist -> head == NULL && arithlist -> tail == NULL)
  {
   return false;
  }

  ListNode * p;
  p = arithlist -> head;

  if(p == ln)
  {
   arithlist -> head = arithlist -> head -> next;
   if(p == arithlist -> tail)
   {
    arithlist -> tail = NULL;
   }
   else
```

```
   {
    arithlist -> head -> prev = NULL;
   }
   free(p);
  }

  else
  {
   while(p != NULL)
   {
    if(p = ln)
    {
     break;
    }
    p = p -> next;
   }

   if(p == NULL)
   {
    return false;
   }

   p -> prev -> next = p -> next;

   if(p -> next != NULL)
   {
    p -> next -> prev = p -> prev;
   }
   else
   {
    arithlist -> tail = p -> prev;
   }
   free(p);
  }
}
#endif
```