```c
/************************************
* EECS2031 - Assignment 1
* Filename: list.c
* Author: Wang, Yang
* Email: infi5959512@hotmail.com
* Login ID: infi999
************************************/


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


List *head, *tail;


/* Display an error message. */

void prtError( char *errMsg )
{
   printf( "%s \n", errMsg );
}


/* Print the content of the list (ignoring the dummy node). */

void prtList()
{
   List *p;
   for ( p = head->next; p != NULL; p = p->next )
        printf( "%4d", p->data );
   printf( "\n");
}



/* Initialize the list. */
/* Create a dummy node to simplify insertion and deletion. */
/* After the list is created, pointers head and tail both point to the dummy node. */
/* Return NULL if a node cannot be created. */
/* Otherwise, return the pointer to the dummy node. */

List *init()
{
   head = ( List * ) malloc( sizeof( List ) );
   if ( head == NULL ) {
      prtError( "Insufficient memory!" );
      return( NULL );
   }
   head->data = -1;
   head->next = NULL;
   tail = head;
   return ( head );
}




/************* DO NOT MODIFY ANYTHING ABOVE THIS LINE, *************/
/************* EXCEPT THE HEADER CONTAINING YOUR INFO **************/

/* Insert a new data element d into the list. */
/* Insert at the front of the list, right behind the dummy node. */
/* Return NULL if a new node cannot be created. */
/* Otherwise, return the pointer to the newly created node. */

List *insertFirst( int d )
{

   /***** ADD YOUR CODE HERE *****/

  List* p = (List*)malloc(sizeof(List));
  if (p == NULL){
    prtError( "Insufficient memory!" );
      return( NULL );
   }
  p->data = d;
  p->next = head->next;
  head->next = p;



}



/* Insert a new data element d into the list. */
/* Insert at the end of the list. */
/* Return NULL if a new node cannot be created. */
/* Otherwise, return the pointer to the newly created node. */

List *insertLast( int d )
{

   /***** ADD YOUR CODE HERE *****/

  List* p = (List*)malloc(sizeof(List));
  if (p == NULL){
    prtError( "Insufficient memory!" );
      return( NULL );
   }

  p->data = d;
  p->next = NULL;
  tail->next = p;
  tail = p;
  return (p);
}



/* Remove the first element of the list, i.e., the node right behind the dummy node. */
/* Return -1 if the list is empty, i.e., containing only the dummy node, */
/* and display error message "Empty list!" on the standard output. */
/* Otherwise, return the data (integer) of the node to be remove. */

int removeFirst()
{

   /***** ADD YOUR CODE HERE *****/
  List *p;
  p = head->next;
  // check empty list
  if (p == NULL){
 prtError( "Empty list!" );
   return -1;
    }
  else{
  // head.next = head.next.next
  head->next = p->next;
    return (p->data);}
}


/* Search the list for an element containing integer k. */
/* If found, return the pointer to that element.  Otherwise, return NULL. */
/* If there is more than one element containing k, return the pointer to the first encountered
element. */

List *search( int k )
{

   /***** ADD YOUR CODE HERE *****/
  List *p;
  p = head-> next;
  for (;p!=NULL;p=p->next){
     if (p->data == k){
       return (p); }}

       return NULL;


}


/*************************** END OF FILE ***************************/
```