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

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "calculate.h"

// DO NOT MODIFY FROM HERE --->>>
const int Operations[] = {'+', '-', '*'};

// return -1 if the word is not an operator
// return 0 if the word contains '+'
// return 1 if the word contains '-'
// return 2 if the word contains '*'
int isOperator(char * word)
{
  int ind;
  int numop = sizeof(Operations) / sizeof(int);
  for (ind = 0; ind < numop; ind ++)
    {
     char *loc = strchr(word, Operations[ind]);
     if (loc != NULL && !isdigit(loc[1]))
        {
          return ind;
        }
    }
  return -1;
}
// <<<--- UNTIL HERE

// ***
// *** You MUST modify the calculate function
// ***
#ifdef TEST_CALCULATE
// if arithlist is NULL, return true
// if arithlist -> head is NULL, return true
// if the input list is invalid, return false
bool calculate(List * arithlist)
{
  ListNode *pntr = arithlist -> head;
  int opreturn; //return of isOperator
  int x;
  int y;
  int new_word;
  //int counter = 0; //increments when an operator is found
  //int counter2 = 0; //increments when an operand is found
```

```
  if (arithlist == NULL)
    {
      return true;
    }
  if ((arithlist -> head) == NULL)
    {
      return true;
    }

  //ListNode * head = arithlist -> head; //checks if head is an
operator
  //if(isOperator(head -> word) != -1)
  //{
   //return false;
  //}

  //ListNode * p = arithlist -> head -> next;
  //if(isOperator(p -> word) != -1) //checks if head -> next is
operator
  //{
   //return false;
  //}

  //ListNode * tail = arithlist -> tail;
  //if(isOperator(tail -> word) == -1) //checks if last node is number
  //{
   //return false;
  //}


  while(pntr != NULL)
  {

   opreturn = isOperator(pntr -> word);
   if(opreturn != -1)
   {
    //counter++;

    if(pntr -> prev == NULL || pntr -> prev -> prev == NULL) //checks
pntr -> prev
    {
     return false;
    }

    else //operator found, now going to operands
    {
     //counter2++;

     x = (int)strtol(pntr -> prev -> word, NULL, 10);
```

```
   y = (int)strtol(pntr -> prev -> prev -> word, NULL, 10);

   if(opreturn == 0) //add
   {
    new_word = x + y;

    deleteNode(arithlist, pntr -> prev);
    deleteNode(arithlist, pntr -> prev);

    sprintf(pntr -> word, "%d\n",  new_word);
   }

   if(opreturn == 1) //subtract
   {
    new_word = y - x;

    deleteNode(arithlist, pntr -> prev);
    deleteNode(arithlist, pntr -> prev);

    sprintf(pntr -> word, "%d\n", new_word);
   }

   if(opreturn == 2) //multiply
   {
    new_word = x * y;

    deleteNode(arithlist, pntr -> prev);
    deleteNode(arithlist, pntr -> prev);

    sprintf(pntr -> word, "%d\n", new_word);
   }

  }//else
 }//if(opreturn != -1)
pntr = pntr -> next;
}//while

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

if(isOperator(arithlist -> head -> word) != -1)
{
 return false;
}
//if(counter == 0)
//{
 //return false;
//}
```

```
  // go through the list until there is only node in the list
  // find the next operator
  // If no operator can be found, return false
  // If an operator is found, find the two previous nodes as operands
  // If cannot find previous two operands, return false
  // If two operands can be found, perform the arithmetic operation
  // Be careful, subtraction is no commutative: 4 2 - means 4 - 2,
  //    not 2 - 4
  // After the operation,
  //     put the result back to the list
  //     remove the two nodes used to store the two operands
  // After going through the entire list and performing the
operations,
  //     the list should have exactly one node left. If this is not
  //     true, return false
  // If the input is valud, return true



  // if more than one node left, return false

  // if the remaining node is an operator, return false

  // if everything is OK, return true
  return true;
}
#endif
```