```
#include "llcpInt.h"
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void SeedRand();
int  BoundedRandomInt(int lowerBound, int upperBound);
void AnsItoArrCalc(const int iArr0[], int used0,
                   int iArr1[], int& used1, int target);
int  ListLengthCheck(Node* head, int whatItShouldBe);
bool match(Node* head, const int procInts[], int procSize);
void ShowArray(const int a[], int size);
void DebugShowCase(int whichCase, int totalCasesToDo,
                   const int caseValues[], int caseSize,
                   int target);

int main()
{
   int testCasesToDo = 990000,
       testCasesDone = 0,
       loSize = 1,
       hiSize = 9,
       loValue = 3,
       hiValue = 7;
   int numInts,
       used0,
       used1,
       intCount,
       newInt,
       target,
       iLenChk1;
   int *iArr0 = 0,
       *iArr1 = 0;
   Node *head1 = 0;

   PromoteTarget(head1, 8533);
   cout << "================================" << endl;
   if (head1 && !head1->link && head1->data == 8533)
   {
      cout << "passed test on empty list" << endl;
      ListClear(head1, 0);
   }
   else
   {
      cout << "failed test on empty list..." << endl;
      ListClear(head1, 0);
      exit(EXIT_FAILURE);
   }

   // SeedRand(); // disabled for reproducible result

   do
   {
      ++testCasesDone;
      numInts = BoundedRandomInt(loSize, hiSize);
      iArr0 = new int [numInts];
      iArr1 = new int [numInts + 1];

      used0 = 0;
      for (intCount = 0; intCount < numInts; ++intCount)
      {
         newInt = BoundedRandomInt(loValue, hiValue);
         iArr0[used0++] = newInt;
         InsertAsTail(head1, newInt);
      }
      target = BoundedRandomInt(loValue, hiValue);
      AnsItoArrCalc(iArr0, used0, iArr1, used1, target);

      //DebugShowCase(testCasesDone, testCasesToDo, iArr0, used0, target);

      PromoteTarget(head1, target);

      iLenChk1 = ListLengthCheck(head1, used1);
      if (iLenChk1 != 0)
      {
         if (iLenChk1 == -1)
         {
            cout << "List node-count error ... too few" << endl;
            cout << "test_case: ";
            ShowArray(iArr0, used0);
            cout << "(target: " << target << ")\n";
            cout << "#expected: " << used1 << endl;
            cout << "#returned: " << FindListLength(head1) << endl;
         }
         else
         {
            cout << "List node-count error ... too many (circular list?)" <<
endl;
            cout << "test_case: ";
            ShowArray(iArr0, used0);
            cout << "(target: " << target << ")\n";
            cout << "#expected: " << used1 << endl;
            cout << "returned # is higher (may be infinite)" << endl;
         }
         exit(EXIT_FAILURE);
      }

      if ( !match(head1, iArr1, used1) )
      {
         cout << "Contents error ... mismatch found in value or order" <<
endl;
         cout << "initial: ";
         ShowArray(iArr0, used0);
         cout << "(target: " << target << ")\n";
         cout << "ought2b: ";
         ShowArray(iArr1, used1);
         cout << "outcome: ";
         ShowAll(cout, head1);
         exit(EXIT_FAILURE);
      }

      if (testCasesDone < 10 || testCasesDone % 30000 == 0)
      {
         cout << "=================================" << endl;
         clog << "testing case " << testCasesDone
              << " of " << testCasesToDo << endl;
         clog << "=================================" << endl;
         // cout << "test case " << testCasesDone
         //      << " of " << testCasesToDo << endl;
         cout << "initial: ";
         ShowArray(iArr0, used0);
         cout << "(target: " << target << ")\n";
         cout << "ought2b: ";
         ShowArray(iArr1, used1);
         cout << "outcome: ";
         ShowAll(cout, head1);
      }

      ListClear(head1, 1);
      delete [] iArr0;
      delete [] iArr1;
      iArr0 = iArr1 = 0;
   }
   while (testCasesDone < testCasesToDo);
   cout << "=================================" << endl;
   cout << "test program terminated normally" << endl;
   cout << "=================================" << endl;

   return EXIT_SUCCESS;
}

/////////////////////////////////////////////////////////////////////
// Function to seed the random number generator
// PRE:  none
// POST: The random number generator has been seeded.
/////////////////////////////////////////////////////////////////////
void SeedRand()
{
   srand( (unsigned) time(NULL) );
}

/////////////////////////////////////////////////////////////////////
// Function to generate a random integer between
// lowerBound and upperBound (inclusive)
// PRE:  lowerBound is a positive integer.
//       upperBound is a positive integer.
//       upperBound is larger than lowerBound
//       The random number generator has been seeded.
// POST: A random integer between lowerBound and upperBound
//       has been returned.
/////////////////////////////////////////////////////////////////////
int BoundedRandomInt(int lowerBound, int upperBound)
{
   return ( rand() % (upperBound - lowerBound + 1) ) + lowerBound;
```

```
}

void AnsItoArrCalc(const int iArr0[], int used0,
                   int iArr1[], int& used1, int target)
{
   used1 = 0;
   for (int i = 0; i < used0; ++i)
      if (iArr0[i] == target) iArr1[used1++] = iArr0[i];
   bool noneMatches = (used1 == 0);
   for (int i = 0; i < used0; ++i)
      if (iArr0[i] != target) iArr1[used1++] = iArr0[i];
   if (noneMatches) iArr1[used1++] = target;
}

/////////////////////////////////////////////////////////////////////
// Function to check # of nodes in list against what it should be
// POST: returns
//          -1 if # of nodes is less than what it should be
//           0 if # of nodes is equal to  what it should be
//           1 if # of nodes is more than what it should be
/////////////////////////////////////////////////////////////////////
int ListLengthCheck(Node* head, int whatItShouldBe)
{
   int length = 0;
   while (head != 0)
   {
      ++length;
      if (length > whatItShouldBe) return 1;
      head = head->link;
   }
   return (length < whatItShouldBe) ? -1 : 0;
}

bool match(Node* head, const int procInts[], int procSize)
{
   int iProc = 0;
   while (head != 0)
   {
      if (iProc == procSize) return false;
      if (head->data != procInts[iProc]) return false;
      ++iProc;
      head = head->link;
   }
   return true;
}

void ShowArray(const int a[], int size)
{
   for (int i = 0; i < size; ++i)
      cout << a[i] << "  ";
   cout << endl;
}

void DebugShowCase(int whichCase, int totalCasesToDo,
                   const int caseValues[], int caseSize,
                   int target)
{
      cout << "=======_BEG_DebugShowCase=======" << endl;
      cout << "test case " << whichCase
           << " of " << totalCasesToDo << endl;
      cout << "given list: ";
      ShowArray(caseValues, caseSize);
      cout << "(target: " << target << ")\n";
      cout << "=======_END_DebugShowCase=======" << endl;
}
```