listTest.c

```
#include "llist.h"

#include <stdio.h>

void testEmpty()
{
  LinkedList* list = ll_new();
  if (!list)
  {
    printf("Failed to create new list\n");
    exit(0);
  }

  if (ll_size(list))
    printf("testEmpty() failed: ll_size() returned non-0\n");
  if (!ll_empty(list))
    printf("testEmpty() failed: ll_empty() returned false\n");

  if (ll_front(list) != NULL)
    printf("testEmpty() failed: ll_front() didn't return NULL\n");
  if (ll_back(list) != NULL)
    printf("testEmpty() failed: ll_back() didn't return NULL\n");

  // nothing to test here, but they should both just work
  ll_pop_front(list);
  ll_pop_back(list);
  if (ll_size(list) != 0)
    printf("testEmpty() failed: "
        "ll_pop_front() or ll_pop_back() reduced list size\n");

  // this should return an invalid iterator
  LinkedIterator it = ll_at(list, 0);
  if (ll_it_valid(&it))
    printf("testEmpty() failed: ll_at(0) returned valid iterator\n");
  it = ll_at(list, 1);
  if (ll_it_valid(&it))
```

```
   printf("testEmpty() failed: ll_at(0) returned valid iterator\n");

  it = ll_it_begin(list);
  if (ll_it_valid(&it))
   printf("testEmpty() failed: ll_it_begin() returned valid iterator\n");

  // all tests done, clean up
  ll_delete(list);
}

int cmp(const void* const a, const void* const b)
{
  return *(int*)a-*(int*)b;
}

LinkedIterator find(LinkedList* const list, int v)
{
  for (LinkedIterator it = ll_it_begin(list); ll_it_valid(&it); ll_it_next(&it))
  {
    if (*(int*)ll_it_data(&it) == v)
      return it;
  }
  LinkedIterator it;
  return it;
}

void testInsertion()
{
  LinkedList* list = ll_new();
  for (int i = 0; i < 5; ++i)
  {
    int* p = malloc(sizeof(int));
    *p = i;
    ll_push_back(list, p);
  }
  for (int i = 6; i < 10; ++i)
  {
    int* p = malloc(sizeof(int));
    *p = i;
    ll_push_front(list, p);
  }

  // test the front and back
  if (*(int*)ll_front(list) != 9)
    printf("testInsertion() falied: ll_front() was not 9\n");
  if (*(int*)ll_back(list) != 4)
```

```
  printf("testInsertion() failed: ll_back() was not 4, got: %i\n",
       *(int*)ll_back(list));

 ll_bsort(list, cmp); // has no return
 // test the front and back again, 0 should be at front, 9 at back
 if (*(int*)ll_front(list) != 0)
  printf("testInsertion() falied: ll_bsort(), front was not 0\n");
 if (*(int*)ll_back(list) != 9)
  printf("testInsertion() failed: ll_bsort(), back was not 9\n");

 {
  LinkedIterator it = find(list, 6);
  int* p = malloc(sizeof(int));
  *p = 5;
  ll_insert(&it, p);
 }
 if (ll_size(list) != 10)
  printf("testInsertion() failed: List size wasn't 10 after insert\n");
 {
  LinkedIterator it = find(list, 5);
  if (!ll_it_valid(&it))
  printf("testInsertion() failed: Failed to find 5 in the list\n");
  ll_it_next(&it);
  free(ll_erase(&it)); // it is now invalid
  if (ll_it_valid(&it))
   printf("testManipulation() failed: "
        "Iterator wasn't invalidated by erase\n");

 }
 for (LinkedIterator it = ll_it_begin(list); ll_it_valid(&it); ll_it_next(&it))
  printf("%i\n", *(int*)ll_it_data(&it));

 ll_clear(list);
 if (!ll_empty(list))
  printf("testManipulation() failed: ll_clear() didn't empty the list\n");

 ll_delete(list);
}

int main()
{
 testEmpty();
 testInsertion();
 return 1;
}
```