```cpp
class ilit ( 
  public: 
  void insert( ilit_item *ptr, int value );
  void insert_all( const ilit &rhs );
  void insert_end( int value );
  void insert_front( int value );
  int remove( ilit_item *ptr );
  int remove( int value );
  void remove_front();
  void remove_all();
  ilit_item *find( int value );
  ilit_item *next_iter();
  ilit_item *init_iter() { return _current = _at_front; }
  void display( ostream & = cout );
  void concat( const ilit & );
  void reverse();
  ilit concat_copy( const ilit & ) const;
  ilit reverse_copy() const;
  int size() { return _size; }
);
#include <cppunit/extensions/HelperMacros.h>
#include "ilist.h"

class ilistTest1 : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE( ilistTest1 );
    CPPUNIT_TEST( testFind );
    CPPUNIT_TEST( testInsert );
    CPPUNIT_TEST( testRemove );
    CPPUNIT_TEST( testRemoveFront );
    CPPUNIT_TEST( testRemoveAll );
    CPPUNIT_TEST_SUITE_END();

protected:
    ilist mylist;

public:
    void setUp();    /* set up context before running a test */
    void tearDown(); /* clean up after the test run */

protected:
    void testFind();
    void testInsert();
    void testRemove();
    void testRemoveFront();
    void testRemoveAll();
};
include "ilistTest1.h"

void ilistTest1::setUp() {
  /*
   * mylist:
   *   ( 20 ) ( 9 8 7 6 5 4 3 2 1 0 0 1 2 3 4 5 6 7 8 9 )
   */
  for (int ix=0; ix<10; ++ix) {
    mylist.insert_front(ix);
    mylist.insert_end(ix);
  }
}

void ilistTest1::tearDown() {
  mylist.remove_all();
}
### istTest1: testFind(), testInsert()

```cpp
void istTest1::testFind() {
    // search for the value 8
    CPPUNIT_ASSERT(0 != mylist.find(8));
    // search for the value 12
    CPPUNIT_ASSERT(0 == mylist.find(12));
}

void istTest1::testInsert() {
    // insert element 1024 at the front, when the iterator = 0
    mylist.insert(0, 1024);
    CPPUNIT_ASSERT_EQUAL(21, mylist.size());
    CPPUNIT_ASSERT_EQUAL(1024, mylist.init_iter()->value());

    // insert element 2048 following the value 8
    ist_item* iter = mylist.find(8);
    mylist.insert(iter, 2048);
    CPPUNIT_ASSERT_EQUAL(22, mylist.size());
    CPPUNIT_ASSERT_EQUAL(2048, iter->next()->value());
}
```
void ilistTest1::testRemove() {
  // remove 2 of the value 8
  CPPUNIT_ASSERT_EQUAL(2, mylist.remove(8));
  CPPUNIT_ASSERT(0 == mylist.find(8));
}

void ilistTest1::testRemoveFront() {
  // remove front element
  int value = mylist.init_iter()->value();
  mylist.remove_front();
  CPPUNIT_ASSERT_EQUAL(19, mylist.size());
  CPPUNIT_ASSERT(mylist.init_iter()->value() != value);
}

void ilistTest1::testRemoveAll() {
  // remove all elements
  mylist.remove_all();
  CPPUNIT_ASSERT_EQUAL(0, mylist.size());
  CPPUNIT_ASSERT(0 == mylist.init_iter());
}
#include <cppunit/extensions/HelperMacros.h>
#include "ilist.h"

class ilistTest2 : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE( ilistTest2 );
    CPPUNIT_TEST( testItemAtEnd );
    CPPUNIT_TEST( testItemAtFront );
    CPPUNIT_TEST( testNoItemsPresent );
    CPPUNIT_TEST( testItemAtFrontAndEnd );
    CPPUNIT_TEST_SUITE_END();

protected:
    ilist mylist;

public:
    void setUp();
    void tearDown();

protected:
    void testItemAtEnd();
    void testItemAtFront();
    void testNoItemsPresent();
    void testItemAtFrontAndEnd();
};
# istTest2 : testItemAtEnd()

```cpp
void istTest2::testItemAtEnd() {
  /*
   * ___________________________________________
   * test #1: items at end
   * ___________________________________________
   *
   * ( 6 ) ( 4 3 2 1 1 1 )
   *
   * Removed 3 of the value 1
   *
   * ( 3 ) ( 4 3 2 )
   *
   */
  myList.insert_front( 1 ); myList.insert_front( 1 ); myList.insert_front( 1 );
  myList.insert_front( 2 ); myList.insert_front( 3 ); myList.insert_front( 4 );
  CPPUNIT_ASSERT_EQUAL(3, myList.remove(1));
  CPPUNIT_ASSERT_EQUAL(3, myList.size());
  ist_list_item *iter = myList.init_iter();
  CPPUNIT_ASSERT_EQUAL(4, iter->value());
  iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(3, iter->value());
  iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(2, iter->value());
  CPPUNIT_ASSERT(0 == iter->next());
}
```
void ilistTest2::testItemAtFront() {
    /*
    * test #2: items at front
    * -------------------------------------
    * ( 3 ) ( 1 1 1 )
    * Removed 3 of the value 1
    * ( 0 ) ( )
    */
    mylist.insert_front( 1 ); mylist.insert_front( 1 ); mylist.insert_front( 1 );
    CPPUNIT_ASSERT_EQUAL(3, mylist.remove(1));
    CPPUNIT_ASSERT_EQUAL(0, mylist.size());
    CPPUNIT_ASSERT(0 == mylist.init_iter());
}
void istTest2::testNoItemsPresent() {
    /*
     * test #3: no items present
     * --------------------------------------------------------
     *
     * ( 3 ) ( 4 2 0 )
     *
     * Removed 0 of the value 1
     *
     * ( 3 ) ( 4 2 0 )
     *
     */
    mylist.insert_front( 0 ); mylist.insert_front( 2 ); mylist.insert_front( 4 );
    CPPUNIT_ASSERT_EQUAL(0, mylist.remove(1));
    CPPUNIT_ASSERT_EQUAL(3, mylist.size());
    ilist_item *iter = mylist.init_iter();
    CPPUNIT_ASSERT_EQUAL(4, iter->value());
    iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(2, iter->value());
    iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(0, iter->value());
    CPPUNIT_ASSERT(0 == iter->next());
void ilistTest2::testItemAtFrontAndEnd() {
    /*
     *  ___________________________________________________________
     *  * test #4: items at front and end
     *  ___________________________________________________________
     *
     *  ( 9 ) ( 1 1 1 4 2 0 1 1 1 )
     *
     * Removed 6 of the value 1
     *
     *  ( 3 ) ( 4 2 0 )
     *
     */
    mylist.insert_front(1); mylist.insert_front(1); mylist.insert_front(1);
    mylist.insert_front(0); mylist.insert_front(2); mylist.insert_front(4);
    mylist.insert_front(1); mylist.insert_front(1); mylist.insert_front(1);

    CPPUNIT_ASSERT_EQUAL(6, mylist.remove(1));
    CPPUNIT_ASSERT_EQUAL(3, mylist.size());
    ilist_item *iter = mylist.init_iter();
    CPPUNIT_ASSERT_EQUAL(4, iter->value());
    iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(2, iter->value());
    iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(0, iter->value());
    CPPUNIT_ASSERT(0 == iter->next());
}
#include <cppunit/extensions/HelperMacros.h>
#include "ilist.h"

class ilistTest3 : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE( ilistTest3 );
    CPPUNIT_TEST( testReverse );
    CPPUNIT_TEST( testConcat );
    CPPUNIT_TEST_SUITE_END();

protected:
    ilist mylist;

public:
    void setUp();
    void tearDown();

protected:
    void testReverse();
    void testConcat();
};
void istTest3::testReverse() {
  /*
   *( 10 )( 9 8 7 6 5 4 3 2 1 0 )
   */
  /* reverse the list */
  /*
   *( 10 )( 0 1 2 3 4 5 6 7 8 9 )
   */
  /*
   */
  for ( int ix = 0; ix < 10; ++ix )
    { mylist.insert_front( ix ); }

  mylist.reverse();

  CPPUNIT_ASSERT_EQUAL(10, mylist.size());
  ist_item *iter = mylist.init_iter();
  CPPUNIT_ASSERT_EQUAL(0, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(1, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(2, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(3, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(4, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(5, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(6, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(7, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(8, iter->value()); iter = iter->next();
  CPPUNIT_ASSERT_EQUAL(9, iter->value());
  CPPUNIT_ASSERT(0 == iter->next());
}
void ilistTest3::testConcat() {
    /*
    * mylist:
    * ( 3 ) ( 3 4 5 )
    * mylist_two:
    * ( 4 ) ( 0 1 1 2 )
    * mylist after concat with mylist_two:
    * ( 7 ) ( 3 4 5 0 1 1 2 )
    */
    mylist.insert_end( 3 ); mylist.insert_end( 4 ); mylist.insert_end( 5 );
    ilist mylist_two;
    mylist_two.insert_end( 0 ); mylist_two.insert_end( 1 );
    mylist_two.insert_end( 1 ); mylist_two.insert_end( 2 );
    mylist.concat(mylist_two);
    CPPUNIT_ASSERT_EQUAL(7, mylist.size());
    ilist_item *iter = mylist.init_iter();
    CPPUNIT_ASSERT_EQUAL(3, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(4, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(5, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(0, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(1, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(1, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(2, iter->value()); iter = iter->next();
    CPPUNIT_ASSERT_EQUAL(0 == iter->next());
}
```cpp
#include <cppunit/ui/text/TestRunner.h>
#include "ilistTest1.h"
#include "ilistTest2.h"
#include "ilistTest3.h"

int main(int argc, char* argv[]) {
    CPPUNIT_NS::TextUi::TestRunner runner;
    runner.addTest(ilistTest1::suite());
    runner.addTest(ilistTest2::suite());
    runner.addTest(ilistTest3::suite());
    bool wasSuccessful = runner.run();

    return wasSuccessful ? 0 : 1;
}
```
Result