```
#include <iostream>
#include <iomanip>

#include "sequence.h"

//#define __GRADING  // Leave this commented out for your tests

#define __CREATE_PRINT
#define __INDEP
#define __PUSH_BACK
#define __PUSH_BACK_EMPTY
#define __POP_BACK
#define __POP_BACK_EMPTY
#define __INSERT
#define __INSERT_INVALID
#define __FRONT
#define __FRONT_EMPTY
#define __BACK
#define __BACK_EMPTY
#define __EMPTY
#define __SIZE
#define __CLEAR
#define __ERASE
#define __ERASE_INVALID
#define __ASSIGNMENT
#define __COPY_CONSTRUCTOR
#define __MEMORY_LEAK_TEST

#define NUM_MEM_TESTS 1000
#define MEM_TEST_SIZE 10

#ifdef __GRADING
#include <fstream>
#define OUTSTREAM os
#else
#define OUTSTREAM cout
#endif


using namespace std;

void testCopyConstructor(Sequence, ostream&);
void memoryLeakTest();

int main()
{

#ifdef __GRADING
        ofstream OUTSTREAM;
        OUTSTREAM.open("eval.txt");
#endif



        // Create a sequence of length four, store some values, and print
        try {
                OUTSTREAM << "Testing sequence creation and printing" << endl;
                OUTSTREAM << "--------------------------------------" << endl;
#ifdef __CREATE_PRINT
                Sequence data(4);
                data[0] = 100;
                data[1] = 200;
                data[2] = 300;
                OUTSTREAM << "Sequence:  " << data << endl;
```

```
                OUTSTREAM << "Should be: <100, 200, 300, ???>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }


#ifdef __GRADING
        OUTSTREAM << "Points (out of 5):            5" << endl << endl;
#endif


        // Test for independent sequences
        try {
                OUTSTREAM << "Testing multiple sequences" << endl;
                OUTSTREAM << "--------------------------" << endl;
#ifdef __INDEP
                Sequence s1(3);
                Sequence s2(3);

                for (int i = 0; i < 3; i++) {
                        s1[i] = i;
                        s2[i] = 100 + i;
                }
                OUTSTREAM << "Sequence1: " << s1 << endl;
                OUTSTREAM << "Sequence2: " << s2 << endl;
                OUTSTREAM << "Should be: <0, 1, 2>" << endl << "           <100, 101, 102>" << endl
<< endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif

        // Test push_back
        try {
                OUTSTREAM << "Testing push_back()" << endl;
                OUTSTREAM << "-------------------" << endl;
#ifdef __PUSH_BACK
                Sequence data(3);
                data[0] = 100;
                data[1] = 200;
                data[2] = 300;
                data.push_back(400);
                data.push_back(500);
                OUTSTREAM << "Sequence:  " << data << endl;
                OUTSTREAM << "Should be: <100, 200, 300, 400, 500>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }
```

```
        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif

        // Test push_back to an empty sequence
        try {
                OUTSTREAM << "Testing push_back() on an empty sequence" << endl;
                OUTSTREAM << "-------------------" << endl;
#ifdef __PUSH_BACK_EMPTY
                Sequence data(0);
                data.push_back(100);
                data.push_back(200);
                data.push_back(300);
                data.push_back(400);
                data.push_back(500);
                OUTSTREAM << "Sequence:  " << data << endl;
                OUTSTREAM << "Should be: <100, 200, 300, 400, 500>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test pop_back
        try {
                OUTSTREAM << "Testing pop_back()" << endl;
                OUTSTREAM << "------------------" << endl;
#ifdef __POP_BACK
                Sequence data(5);
                for (int i = 0; i < 5; i++) {
                        data[i] = (i + 1) * 100;
                }
                data.pop_back();
                data.pop_back();
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <100, 200, 300>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif

        // Test pop_back on empty sequence
        try {
```

```
                OUTSTREAM << "Testing pop_back() on an empty sequence" << endl;
                OUTSTREAM << "------------------" << endl;
#ifdef __POP_BACK_EMPTY
                Sequence data(3);
                for (int i = 0; i < 3; i++) {
                        data[i] = (i + 1) * 100;
                }
                data.pop_back();
                data.pop_back();
                data.pop_back();
                data.pop_back();
                OUTSTREAM << "ERROR: Pop_back() DID NOT throw an exception" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "CORRECT: Threw exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test insert()
        try {
                OUTSTREAM << "Testing insert()" << endl;
                OUTSTREAM << "------------------" << endl;
#ifdef __INSERT
                Sequence data(5);

                for (int i = 0; i < 5; i++) {
                        data[i] = (i + 1) * 100;
                }

                data.insert(3, 999);
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <100, 200, 300, 999, 400, 500>" << endl << endl;

                data.insert(0, 888);
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <888, 100, 200, 300, 999, 400, 500>" << endl << endl;

                data.insert(6, 777);
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <888, 100, 200, 300, 999, 400, 777, 500>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif

        // Test insert on invalid index
        try {
                OUTSTREAM << "Testing insert() on an invalid index" << endl;
```

```
                OUTSTREAM << "------------------" << endl;
#ifdef __INSERT_INVALID
                Sequence data(3);
                for (int i = 0; i < 3; i++) {
                        data[i] = (i + 1) * 100;
                }
                data.insert(5, 555);
                OUTSTREAM << "ERROR: Pop_back() DID NOT throw an exception" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "CORRECT: Threw exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test front()
        try {
                OUTSTREAM << "Testing front()" << endl;
                OUTSTREAM << "---------------" << endl;
#ifdef __FRONT
                Sequence data(3);

                for (int i = 0; i < 3; i++) {
                        data[i] = (i + 1) * 100;
                }

                OUTSTREAM << "Front:     " << data.front() << endl;
                OUTSTREAM << "Sequence:  " << data << endl;
                OUTSTREAM << "Should be: 100" << endl;
                OUTSTREAM << "           <100, 200, 300>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 2):            2" << endl << endl;
#endif

        // Test front() on empty sequence
        try {
                OUTSTREAM << "Testing front() on an empty sequence" << endl;
                OUTSTREAM << "-----------------------------------" << endl;
#ifdef __FRONT_EMPTY
                Sequence data(0);
                int result = data.front();
                OUTSTREAM << "ERROR: front() DID NOT throw an exception" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
```

```
        {
                OUTSTREAM << "CORRECT: Threw exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test back()
        try {
                OUTSTREAM << "Testing back()" << endl;
                OUTSTREAM << "---------------" << endl;
#ifdef __BACK
                Sequence data(3);

                for (int i = 0; i < 3; i++) {
                        data[i] = (i + 1) * 100;
                }

                OUTSTREAM << "Back:      " << data.back() << endl;
                OUTSTREAM << "Sequence:  " << data << endl;
                OUTSTREAM << "Should be: 300" << endl;
                OUTSTREAM << "           <100, 200, 300>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 2):            2" << endl << endl;
#endif

        // Test back() on empty sequence
        try {
                OUTSTREAM << "Testing back() on an empty sequence" << endl;
                OUTSTREAM << "-----------------------------------" << endl;
#ifdef __BACK_EMPTY
                Sequence data(0);
                int result = data.back();
                OUTSTREAM << "ERROR: back() DID NOT throw an exception" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
                OUTSTREAM << "CORRECT: Threw exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test empty()
        try {
                OUTSTREAM << "Testing empty()" << endl;
                OUTSTREAM << "---------------" << endl;
```

```
#ifdef __EMPTY
                Sequence empty_sequence(0);
                Sequence nonempty_sequence(1);

                OUTSTREAM << "Empty sequence, empty returns: " << empty_sequence.empty() << endl;
                OUTSTREAM << "Non-empty sequence, empty returns: " << nonempty_sequence.empty() <<
endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test size()
        try {
                OUTSTREAM << "Testing size()" << endl;
                OUTSTREAM << "---------------" << endl;
#ifdef __SIZE
                Sequence data(7);
                Sequence empty_sequence(0);

                OUTSTREAM << "Sequence length 7, size returned: " << data.size() << endl;
                OUTSTREAM << "Empty sequence, size returned: " << empty_sequence.size() << endl <<
endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 2):            2" << endl << endl;
#endif

        // Test clear()
        try {
                OUTSTREAM << "Testing clear()" << endl;
                OUTSTREAM << "---------------" << endl;
#ifdef __CLEAR
                Sequence data(5);

                for (int i = 0; i < 5; i++) {
                        data[i] = (i + 1) * 100;
                }

                data.clear();
                OUTSTREAM << "Sequence cleared, empty returned: " << data.empty() << endl;
                OUTSTREAM << "Size returned:  " << data.size() << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }
```

```
        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 2):            2" << endl << endl;
#endif

        // Test erase
        try {
                OUTSTREAM << "Testing erase()" << endl;
                OUTSTREAM << "------------------" << endl;
#ifdef __ERASE
                Sequence data(10);
                for (int i = 0; i < 10; i++) {
                        data[i] = (i + 1) * 100;
                }
                data.erase(3, 4);
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <100, 200, 300, 800, 900, 1000>" << endl << endl;

                data.erase(4, 2);
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <100, 200, 300, 800>" << endl << endl;

                data.erase(0, 2);
                OUTSTREAM << "Sequence:   " << data << endl;
                OUTSTREAM << "Should be:  <300, 800>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif

        // Test erase with invalid parameters
        try {
                OUTSTREAM << "Testing erase() on invalid parameters" << endl;
                OUTSTREAM << "------------------" << endl;
#ifdef __ERASE_INVALID
                Sequence data(5);
                for (int i = 0; i < 5; i++) {
                        data[i] = (i + 1) * 100;
                }
                data.erase(3, 5);
                OUTSTREAM << "ERROR: erase() DID NOT throw an exception" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif
        }

        catch (exception& e)
        {
                OUTSTREAM << "CORRECT: Threw exception: " << e.what() << endl << endl;
        }
```

```
#ifdef __GRADING
        OUTSTREAM << "Points (out of 1):            1" << endl << endl;
#endif

        // Test assignment (=) operator
        try {
                OUTSTREAM << "Testing assignment (=) operator" << endl;
                OUTSTREAM << "-------------------" << endl;
#ifdef __ASSIGNMENT
                Sequence data1(5);
                Sequence data2(0);

                for (int i = 0; i < 5; i++) {
                        data1[i] = (i + 1) * 100;
                }

                data2 = data1;

                data2[0] = 1;
                data2[1] = 2;

                OUTSTREAM << "Data1:      " << data1 << endl;
                OUTSTREAM << "Data2:      " << data2 << endl;
                OUTSTREAM << "Should be:  <100, 200, 300, 400, 500>" << endl;
                OUTSTREAM << "             <1, 2, 300, 400, 500>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif

        // Test copy constructor
        try {
                OUTSTREAM << "Testing copy constructor" << endl;
                OUTSTREAM << "-------------------" << endl;
#ifdef __COPY_CONSTRUCTOR
                Sequence data(5);

                for (int i = 0; i < 5; i++) {
                        data[i] = (i + 1) * 100;
                }

                testCopyConstructor(data, OUTSTREAM);

                OUTSTREAM << "Original Sequence:      " << data << endl;
                OUTSTREAM << "Should be:              <1, 200, 300, 400, 500>" << endl;
                OUTSTREAM << "                        <100, 200, 300, 400, 500>" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
```

```
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif


        // Test for memory leaks
        try {
                OUTSTREAM << "Testing for memory leaks" << endl;
                OUTSTREAM << "------------------------" << endl;
#ifdef __MEMORY_LEAK_TEST
                cout << "Pre-memory leak test..." << endl;
                system("pause");
                for (int i = 0; i < NUM_MEM_TESTS; i++) {
                        memoryLeakTest();
                }
                cout << "Post-memory leak test..." << endl;
                system("pause");

                OUTSTREAM << "No memory leak found" << endl;
                OUTSTREAM << "*** Error:  memory leak found" << endl << endl;
#else
                OUTSTREAM << "*** UNHANDLED CRASH DURING TESTING ***" << endl << endl;
#endif

        }

        catch (exception& e)
        {
                OUTSTREAM << "Exception: " << e.what() << endl << endl;
        }

#ifdef __GRADING
        OUTSTREAM << "Points (out of 3):            3" << endl << endl;
#endif


#ifdef __GRADING
        OUTSTREAM << "Programming style and documentation" << endl;
        OUTSTREAM << "-----------------------------------" << endl;
        OUTSTREAM << "Points (out of 5):            5" << endl << endl;
        OUTSTREAM << "TOTAL POINTS (out of 50):    50" << endl;
        OUTSTREAM.close();
#endif

} // END OF MAIN

void memoryLeakTest() {
        Sequence s(MEM_TEST_SIZE);
        for (int i = 0; i < MEM_TEST_SIZE; i++) {
                s[i] = i;
        }
}

void testCopyConstructor(Sequence s, ostream &os)
{
        s[0] = 1;
        os << "Copied Sequence:        " << s << endl;
}
```