Demonstrate "Smart Pointers".

This example demonstrates the use of "smart pointers", that keep up with how many references there are to each allocated memory block, and only free the space when the reference count goes to zero.

#include <iostream>
#include <string>

using namespace std;

class SPointer
{
public:
    SPointer(char*);
    SPointer(const SPointer&); // Need a copy constructor
    ~SPointer(); // Need a destructor
    SPointer& operator=(const SPointer& rhs); // Need an assignment operator

    char Get(int i); // Get the "i’th" character from the array
    void Set(int i, char c); // Set the "i’th" character to the char ‘c’
    void Set(const char* s); // Set the string to s
    void Print(const string&); // Print the string

private:
    char* pointer; // This is the "shared" pointer
    int* refCount; // This keeps up with how many references there are
    int lth;

public:
    static int allocCount; // Debug..track count of alloc/deletes
    static int deleteCount;
};

// Constructor
SPointer::SPointer(char* s)
: lth(strlen(s) + 1)
{
    pointer = new char[lth]; // Allocate the dynamic memory
    strcpy(pointer, s);
    refCount = new int(1); // Create the reference count variable, set to 1
    allocCount++; // For debug, count the allocations
}

// Copy constructor
SPointer::SPointer(const SPointer& c)
: pointer(c.pointer), refCount(c.refCount), lth(c.lth)
{
    (*refCount)++; // Increment the reference count

Program smartpointers.cc
// Destructor
SPointer::~SPointer()
{
    (*refCount)--; // Decrement the reference count
    if (*refCount == 0)
        // This is the last reference, delete
        delete [] pointer;
    delete refCount;
    deleteCount++; // For debug, count the deletions
}

// Assignment operator
SPointer& SPointer::operator=(const SPointer& rhs)
{
    if (pointer != rhs.pointer)
        // not self assignment
        (*refCount)--;
    if (*refCount == 0)
        // Last reference to my string, delete
        delete [] pointer;
    delete refCount;
    deleteCount++; // For debug, count the deletions
    pointer = rhs.pointer;
    refCount = rhs.refCount;
    lth = rhs.lth;
    (*refCount)++; // Count this reference
    return *this;
}

// Get and Set functions
char SPointer::Get(int i)
{
    if (i < lth) return pointer[i];
    return '0'; // Out of range, just return 0
}

void SPointer::Set(int i, char c) // Set the "i'th" character to the char 'c'
{
    // Set the new value. However, we must make a copy of the data
    // This is called "Copy on Write" semantics
    if (i >= lth) return; // Out of range
    if (*refCount > 1)
        // If this is not the only reference, we need to realloc and copy
        (*refCount)--; // Decrement reference count
    char* newPointer = new char[lth]; // Get a new memory array
    refCount = new int(1); // Get a new reference count = 1
    memcpy(newPointer, pointer, lth); // Copy the data
    pointer = newPointer; // And set the new pointer
    allocCount++; // Count the allocation
}
pointer[i] = c;

Program smartpointers.cc (continued)
void SPointer::Set(const char* s) // Set the string to s
{ // Length of s must be less than or equal to lth
  if (*refCount > 1)
  { // If this is not the only reference, we need to realloc and copy
    (*refCount)--; // Decrement reference count
    char* newPointer = new char[lth]; // Get a new memory array
    refCount = new int(1); // Get a new reference count int
    pointer = newPointer; // And set the new pointer
    allocCount++; // Count the allocation
  }
  strncpy(pointer, s, lth); // Copy, but no more then "length"
}

void SPointer::Print(const string& prompt)
{
  cout << prompt << " pointer is " << (void*)pointer
       << " string is " << string(pointer)
       << "\", refCount " << *refCount
       << endl;
}

int SPointer::allocCount = 0;
int SPointer::deleteCount = 0;

void Sub()
{ // Make a subroutine that tests SPointers.
  // We use a subroutine so that all SPointers will go out of scope
  // on exit, so we can verify the allocCount and deleteCount are the
  // same.
  SPointer sp1("This is a test");
  SPointer sp2(sp1); // Copy constructor, sp2 shares the pointer with sp1
  SPointer sp3("ShortString");
  sp3 = sp1; // Assignment operator, sp3 also shares the pointer with sp1
  sp1.Print("sp1");
  sp2.Print("sp2");
  sp3.Print("sp3");
  cout << endl;
  // Now change sp1, and notice that sp2/sp3 don’t change
  sp1.Set("Another test");
  sp1.Print("sp1");
  sp2.Print("sp2");
  sp3.Print("sp3");
  cout << endl;
  // Change sp2, see that sp1 and sp3 are unchanged
  sp2.Set(0, 'K');
  sp1.Print("sp1");
  sp2.Print("sp2");
  sp3.Print("sp3");
  cout << endl;
}

Program smartpointers.cc (continued)
int main()
{
    Sub();
    cout << "AllocCount is " << SPointer::allocCount << endl;
    cout << "DeleteCount is " << SPointer::deleteCount << endl;
}

// The output from this program is:

// sp1 pointer is 0xe2400 string is "This is a test", refCount 3
// sp2 pointer is 0xe2400 string is "This is a test", refCount 3
// sp3 pointer is 0xe2400 string is "This is a test", refCount 3

// sp1 pointer is 0xe2720 string is "Another test", refCount 1
// sp2 pointer is 0xe2400 string is "This is a test", refCount 2
// sp3 pointer is 0xe2400 string is "This is a test", refCount 2

// sp1 pointer is 0xe2720 string is "Another test", refCount 1
// sp2 pointer is 0xe2770 string is "Khis is a test", refCount 1
// sp3 pointer is 0xe2400 string is "This is a test", refCount 1

// AllocCount is 4
// DeleteCount is 4