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; // Length of shared memory

    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
}

// Destructor
SPointer::~SPointer()
{

}
// Decrement the reference count
if (*refCount == 0)
    { // Last reference to my string, 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'
{
    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;
}

void SPointer::Set(const char* s) // Set the string to s
{
    if (strlen(s) > 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
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < lth; j++)
        pointer[j] = s[j];
}

// Copy the string from s to my array
void SPointer::Copy(const char* s)
{
    if (*refCount > 1)
        { // If this is not the only reference, we need to realloc and copy
            (*refCount)--; // Decrement reference count
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < lth; j++)
        pointer[j] = s[j];
}

// Copy the string from s to my array
void SPointer::Copy(const SPointer& rhs)
{
    if (*refCount > 1)
        { // If this is not the only reference, we need to realloc and copy
            (*refCount)--; // Decrement reference count
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < lth; j++)
        pointer[j] = rhs.pointer[j];
}

// Copy the string from s to my array
void SPointer::Copy(const char* s, int l)
{
    if (*refCount > 1)
        { // If this is not the only reference, we need to realloc and copy
            (*refCount)--; // Decrement reference count
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < l; j++)
        pointer[j] = s[j];
}

// Copy the string from s to my array
void SPointer::Copy(const SPointer& rhs, int l)
{
    if (*refCount > 1)
        { // If this is not the only reference, we need to realloc and copy
            (*refCount)--; // Decrement reference count
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < l; j++)
        pointer[j] = rhs.pointer[j];
}

// Copy the string from s to my array
void SPointer::Copy(const char* s, int l, int n)
{
    if (*refCount > 1)
        { // If this is not the only reference, we need to realloc and copy
            (*refCount)--; // Decrement reference count
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < l; j++)
        pointer[j] = s[j];
}

// Copy the string from s to my array
void SPointer::Copy(const SPointer& rhs, int l, int n)
{
    if (*refCount > 1)
        { // If this is not the only reference, we need to realloc and copy
            (*refCount)--; // Decrement reference count
            refCount = new int(1); // Get a new reference count = 1
            pointer = newPointer; // And set the new pointer
            allocCount++; // Count the allocation
        }
    for (int j = 0; j < l; j++)
        pointer[j] = rhs.pointer[j];
}
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;
}

// Test program
int main()
{
Program smartpointers.cc (continued)
169 Sub();
170 cout << "AllocCount is " << SPointer::allocCount << endl;
171 cout << "DeleteCount is " << SPointer::deleteCount << endl;
172 }
173
174 // The output from this program is:
175 ///
176 // sp1 pointer is 0xe2400 string is "This is a test", refCount 3
177 // sp2 pointer is 0xe2400 string is "This is a test", refCount 3
178 // sp3 pointer is 0xe2400 string is "This is a test", refCount 3
179
180 // sp1 pointer is 0xe2720 string is "Another test", refCount 1
181 // sp2 pointer is 0xe2400 string is "This is a test", refCount 2
182 // sp3 pointer is 0xe2400 string is "This is a test", refCount 2
183
184 // sp1 pointer is 0xe2720 string is "Another test", refCount 1
185 // sp2 pointer is 0xe2770 string is "Khis is a test", refCount 1
186 // sp3 pointer is 0xe2400 string is "This is a test", refCount 1
187
188 // AllocCount is 4
189 // DeleteCount is 4
190
191
192
193
194
195
196
197
198
199
200

Program smartpointers.cc (continued)