

The SteveString Class

```
// SteveString.h
// Specifies a class that allows the
// user to use resizable arrays of chars.
#ifndef _STEVESTRING_H_
#define _STEVESTRING_H_

class SteveString {
public:
    // our constructor
    SteveString ( );

    // NEW: we want to have a copy constructor
    // so we can initialize SteveStrings with copies of
    // other SteveStrings
    SteveString ( SteveString & );

    // add a char to our SteveString, resizing
    // if there isn't enough room.
    void addItem( char x );

    // our destructor
    ~SteveString ( );

    // NEW: we now overload operator+, so
    // that—like strings—we can concatenate using
    // the + operator in SteveStrings.
    SteveString& operator+( SteveString & );

private:
    // current number of items in the SteveString
    int currentSize;

    // current size of the SteveString
    int totalCapacity;

    // our data, a pointer because we are going to
    // dynamically allocate it.
    char* data;
};
#endif
```

Implementation of the SteveString Class:

```
#include "SteveString.h"
```

```
SteveString::SteveString( ){  
    totalCapacity = 10; currentSize = 0;  
    data = new char[totalCapacity];  
}
```

```
SteveString::SteveString( SteveString &other ){  
    // get other's capacity and size  
    totalCapacity = other.totalCapacity;  
    currentSize = other.currentSize;  
    // declare our data array  
    data = new char[totalCapacity];  
    // now copy other's data. This is a DEEP copy  
    for (int i = 0; i < totalCapacity; i++)  
        data[i] = other.data[i];  
    // we're done! We now have a COMPLETE  
    // copy of the data structure!  
}
```

```
void SteveString::addItem( char x ) {  
    // is there room?  
    if ((currentSize + 1) < totalCapacity) {  
        currentSize++;  
        data[currentSize] = x;  
        return;  
    }  
    // there isn't room, so we need to resize the array.  
    else {  
        totalCapacity = totalCapacity * 2;  
        char * newData = new char[totalCapacity];  
        for (int i = 0; i < currentSize; i++)  
            newData[i] = data[i];  
        delete [] data;  
        data = newData;  
        currentSize++;  
        data[currentSize] = x;  
        return;  
    }  
}
```

Implementation of the SteveString Class (continued):

```
SteveString& SteveString::operator+( SteveString &other ){

    // first, we need to make a new array with the right
    // amount of space to hold both strings, and get some

    // new size values.
    int newCapacity = (totalCapacity + other.totalCapacity);
    int newSize = (currentSize + other.currentSize);
    char* newData = new char[newCapacity];

    // now, we need copy the first part into the new array
    for (int i = 0; i < currentSize; i++)
        newData[i] = data[i];

    // and now, the second part
    for (int x = currentSize; x < newSize; x++)
        newData[x] = other.data[(x - currentSize)];

    // now we can delete our current data and reassign the
    // new data.
    delete [] data;
    data = newData;
    totalCapacity = newCapacity;
    currentSize = newSize;

    // new, we return a reference to the SteveString, since
    // the + operator needs to return something.
    return *this;

    // that's it! We're done!!
}

SteveString::~SteveString( ){
    delete [] data;
}
```

Some review questions to ponder. . .

- What's the difference between deep and shallow copy?
- In what order, when using instances of classes inside other classes, are constructors called? Destructors?
- What does const do? What are some different ways to use it? What are the differences between these ways?
- What's the difference between static, dynamic, and automatic memory? Can you give an example of each?
- What is the 'this' keyword? How is it used? Why do we have it?
- What is the difference between an alias, or reference variable, and a pointer variable?
- How many difference constructors can we have in a class? How does the compiler know which one to call?
- What is a destructor? What does it do? Why is it needed?
- What does 'operator overloading' mean? Why do we do it?
- What is a 'memory leak'? Give an example. What is a 'dangling pointer'? Give an example of that too.
- How many licks does it take to get to the center of a tootsie-roll pop? Write an algorithm that calculates this. (ok, so maybe you don't need to know this one. . .)
- What happens when you don't specify a constructor and/or deconstructor in a class?
- What is multiple inclusion? Describe the fix we have in order to solve the problem.