Destructor

When dealing with dynamically allocated memory, you need to have a **destructor** in your program. The dynamic data will not be formally deleted until you delete it with the keyword **delete**. Without a destructor, you are liable to have memory leaks. The destructor is automatically invoked when an object of the class goes out of scope. Here is an example of a class with a destructor for a dynamic array:

```
class Dynamic
{
    public:
        //Constructors
        ~Dynamic();
        //Member Functions
    private:
        int *array;
        //Other member variables
};

Dynamic::~Dynamic()
{   //Formally delete dynamic data
    delete [] array;
}
```

Notice that the destructor begins with a tilde **~** and is followed by the name of the class. There is **no** return type.

You are able to do more cleanup in the destructor if you would like, such as resetting all variables to their default values.

Please keep in mind that temporary objects with dynamically allocated data created in global and member functions will be deleted as soon as that function terminates via the destructor. If you are returning a temporary object, it is very important to utilize the **Rule of Three**. This is to properly make a deep copy of the dynamically allocated data. Remember the **Rule of Three** states that if you need a destructor, then you need to include a copy constructor and an assignment operator as well (see the **Rule of Three** section).

Here is how to delete a two-dimensional array (see **MultiArray.cpp**):

```
Dynamic::~Dynamic()
{   //Delete array of int pointers one-by-one
    for(int i = 0; i < NUMROWS; ++i)
        delete [] array[i];
    //Delete the pointer to the 2D-array
    delete [] array;
}
```