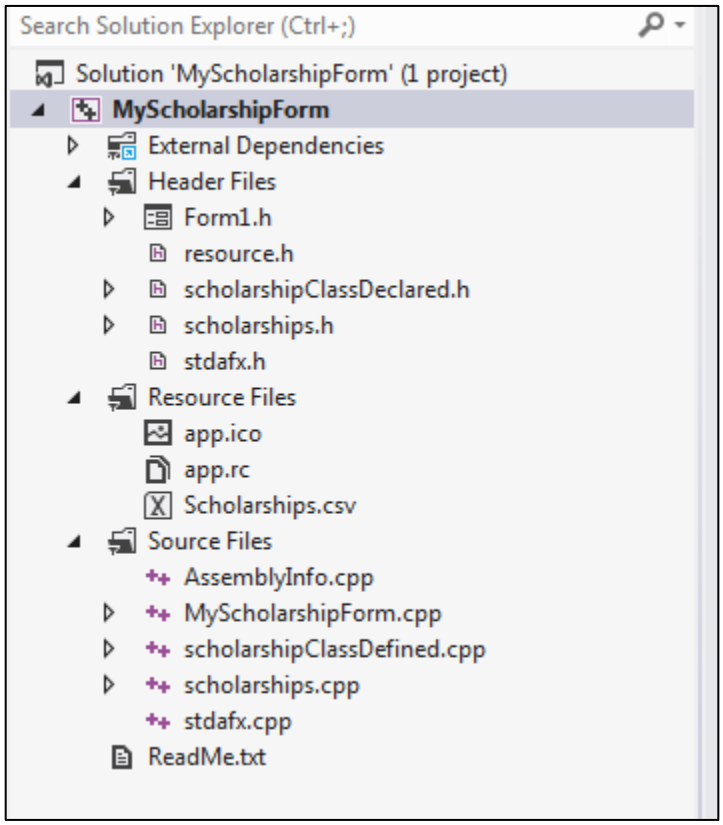
More About Classes Tutorial Part 1 – MyScolarshipForm

This tutorial demonstrates more about classes and objects. It uses the Scholarship vector array of objects that you have seen in past tutorials. DO NOT USE a previous project. In this tutorial, the project is already made. You will simply download it and open the solution. DELETE the OLD project if you have to so as to avoid folder name conflicts within the same directory.

NOTE: If you find errors when working this tutorial, do NOT start adding or deleting your *own* code to fix it. This tutorial *does* work when done correctly. If you find a typing error or run into trouble, do not hesitate to ask the instructor for clarification or assistance.

1. Download the More About Classes Tutorial Part 1 ZIP file from Blackboard>Lesson 7 and save to your computer.
2. Using Windows Explorer, COPY THE ENTIRE PROJECT from the ZIP file into onto your Desktop for easy access.
3. Open the project folder (*MyScholarshipForm)* and then open the *MyScholarshipForm.sln* in Visual Studio 2012. Your project's Solution Explorer panel should look like the one below.



4. **Run** the program to make sure it is set up correctly.

5. With the program running, click the **First** and **Last** buttons and see the error message pop up. Exit the program. Naturally, you don't need this message here since the user intended to see the first and last records. Add the two *if()* statements to the *fillFormFields()* method as shown below. This way, the messages will only show if the Previous or Next buttons trigger the method.

```
void fillFormFields(int p)
{
    static int i = 0;  // Keeps track of which record we are on
    i += p;            // changes the current record

    if(i < 0)  // Is the index within bounds
    {
        i = 0; // reset to beginning of array
        if(p == -1)
            MessageBox::Show("Cannot go past beginning of array.
                             "Notification", MessageBoxButtons::C
    }
    if(i > int((*s).size()-1))  // Is the index within bounds
    {
        i = int((*s).size()-1); // reset to end of array
        if(p == 1)
            MessageBox::Show("Cannot go past end of array. Showir
                             "Notification", MessageBoxButtons::C
    }
```

6. **Run** the program to make sure the error only appears when the Next or Previous buttons try to go past the end of the array. Then Stop the program.

Static Class Member Variables

Suppose you want to save processing time by having the *Scholarship* class keep track of how many objects are in memory. This way, you would not have to execute the s->size() command every time you need to know how many there are. You can do this by making a new member variable that is static. ALL the objects in memory share this variable. So, if one object changes the value it is then changed for ALL objects.

7. Open the *Scholarship* Class Declaration and add the following member variable as highlighted below.

```
class Scholarship
{
private:
    string ID;
    int    Amount;
    string Type;
    string Length;
    string DateStarts;
    string Lname;
    string Fname;
    static int numScholarships = 0;
```

8. Hover over the error underscore and read the error message. You cannot give values to static members of classes because no objects have been made in memory to store the value. This only happens when the first object is made.
9. Delete the " = 0" from the statement you just added to get rid of the error.
10. Go to the top of the *Scholarship* class definition file and add the following line of code to initialize the new static member "once" *before* any objects are made.

```
// scholarshipClassDefined.cpp
// Member Function definitions for the Scholarship class
#include "stdafx.h"
#include "scholarshipClassDeclared.h"

int Scholarship::numScholarships = 0;

Scholarship::Scholarship()                  // default constructor
{
    ID = "";
    Amount = 0;
    Type = "";
    Length = "";
    DateStarts = "";
    Lname = "";
    Fname = "";

}
```

11. To add and subtract as from this new member variable, you *add* 1 to it each time a *Scholarship* object is made and you *subtract* when one is destroyed (should you delete one during later development of the program).
12. **Add** a "get" method to your class to retrieve the private *numScholarships* member variable value when needed. *There is no need for a "set" method as this is handled inside the class.*
13. **Add** the following statement to BOTH constructors.

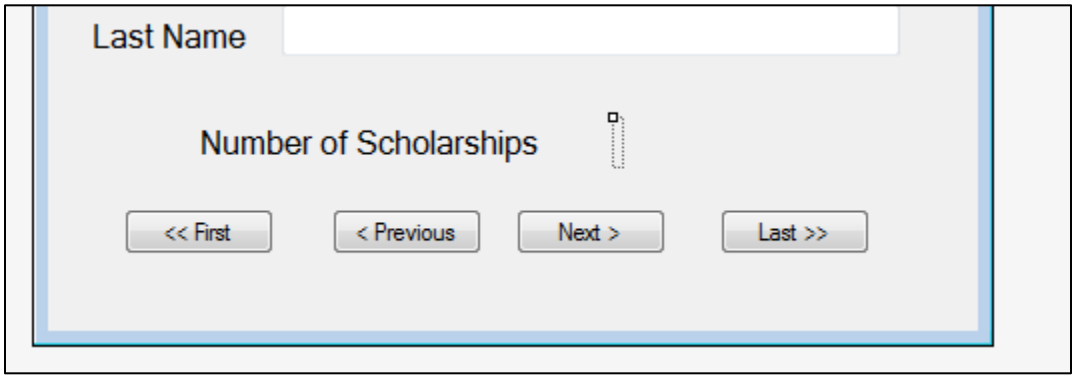
```
numScholarships++;
```

14. **Add** the following *destructor* to the Scholarships class.

```
Scholarship::~Scholarship()                 // default destructor
{
    numScholarships--;
}
```

15. **Run** the program to make sure everything still works ok.

16. To use the new member, open Form1 in design view and add 2 labels as shown below. The text in one should be "Number of Scholarships" and the other label should have no text.



17. Open the Properties panel for the *empty* label and name it "**labelNumS".**
18. **Add** this line of code to the bottom of the *fillFormFields()* method.

```
labelNumS->Text = gcnew String((*s)[i].getNumScholarships().ToString());
```

19. **Run** the program. How many Scholarship objects are there? Is this correct? Do you know why or why not?

    The reason the number of objects in memory is incorrect is because there is one more constructor type that you have not overridden yet. That being the *copy constructor.* All classes have a default *copy constructor* that simply creates a "temporary" object in memory without using the *default constructor*, copies all its values to the new object and then calls the *destructor.* You program uses the *default copy constructor* several times when reading the file and loading the array. Therefore, only the *numScholarships--* is being executed with no ++ since this code is not in the *copy constructor.* You need to add a copy constructor that adds 1 to the *numScholarships* variable each time it is used.
20. Add the below *default copy constructor* just above the *destructor*. In the class definition.

```
    numScholarships++;

}
// default copy constructor
Scholarship::Scholarship(Scholarship &obj)
{
    ID = obj.ID;
    Amount = obj.Amount;
    Type = obj.Type;
    Length = obj.Length;
    DateStarts = obj.DateStarts;
    Lname = obj.Lname;
    Fname = obj.Fname;
    obj.numScholarships++;
}
Scholarship::~Scholarship()
{
    numScholarships--;
}
```

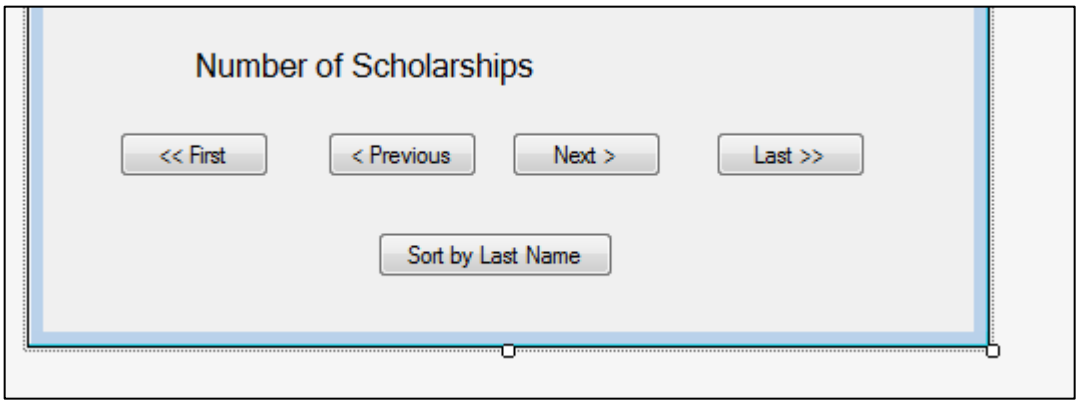
For more information on copy constructors, read this section of the textbook. It is also relevant to initializing pointers and copying pointers to new objects. The book gives an example on how you must override the *default copy constructor* if you initialize a pointer in your default constructor. Since the *default copy constructor* does a "member-wise" straight copy of values, all it copies is the "address" where a pointer is pointing to. YOU must write the copy that actually copies the value of what is pointed to into a new location and give the address of the new location to the "copied to" object's pointer variable. The textbook has a good example of how you should code this.

21. **Run** the program and see if this fixed the problem.

Operator Overloading

In the *bubbleSort*() function from a previous tutorial, you made an "external" function to determine how to sort the array of *Scholarship* objects. You can add functionality to your *Scholarship* class to handle different ways of sorting from *inside* the class. Wouldn't it be easier just to write code in a sort function like "if(scholar1 < scholar2)" ? Unfortunately, the < *operator* does not know how to handle objects of type Scholarship. You can teach it to. This is called "operator overloading" which is similar to function or method overloading.

22. **Open** *Form1* in design view and make the form a little taller to make room for a new button at the bottom.
23. **Add** a new button to *Form1* just below the Previous and Next buttons (see the previous tutorial on Windows Forms if you cannot remember how to do this).
24. **Open** the *properties* panel for the new button and set the *Name* property to "**btnSortByLastName"** and *Text* property to "**Sort by Last Name." Resize** the button so that the entire text appears. The bottom of your form should look similar to this.



25. **Double-click** the new button to create a new "click" handler method, which will then open for editing.

26. Add the highlighted bubble sort code below to the new “click” method.

```
private: System::Void btnSortByLastName_Click(System::Object^  sender, System::

            Scholarship temp;    // Holds Scholarship object
            bool swap;
            do
            {  swap = false;
                for (int count = 0; count < long((*s).size() - 1); count++)
                {
                    if (s[count] > s[count + 1])
                    {
                        temp = (*s)[count];
                        (*s)[count] = (*s)[count + 1];
                        (*s)[count + 1] = temp;
                        swap = true;
                    }
                }
            } while (swap);
        }
```

27. Try running the program. You should get a build error because the *< operator* does NOT know how to handle operands of type Scholarship objects. Press the F4 key on the keyboard to see the error message.
28. **Add** the following method to your Scholarship class.

```
bool Scholarship::operator<(const Scholarship &right)
{
    if(this->Lname < right.Lname)
        return true;
    else
        return false;
}
```

29. **Run** the program and test the new button out. Click the new button, and then scroll through the records. They should be in descending order.
30. **YOUR TURN:** Add another overloaded operator method for the > operator. Then add a new button that sorts ascending by last name. Add the word “descending” and “ascending” to the two sort buttons so the user knows which is which.
31. **Now, Overload** the “+” and “-“ operators so that you can add to and subtract from the “Amount” of a given Scholarship object when an integer value is added or subtracted on the right of the object. HINT: The method signature for the “+” would be: **void operator+(int);**