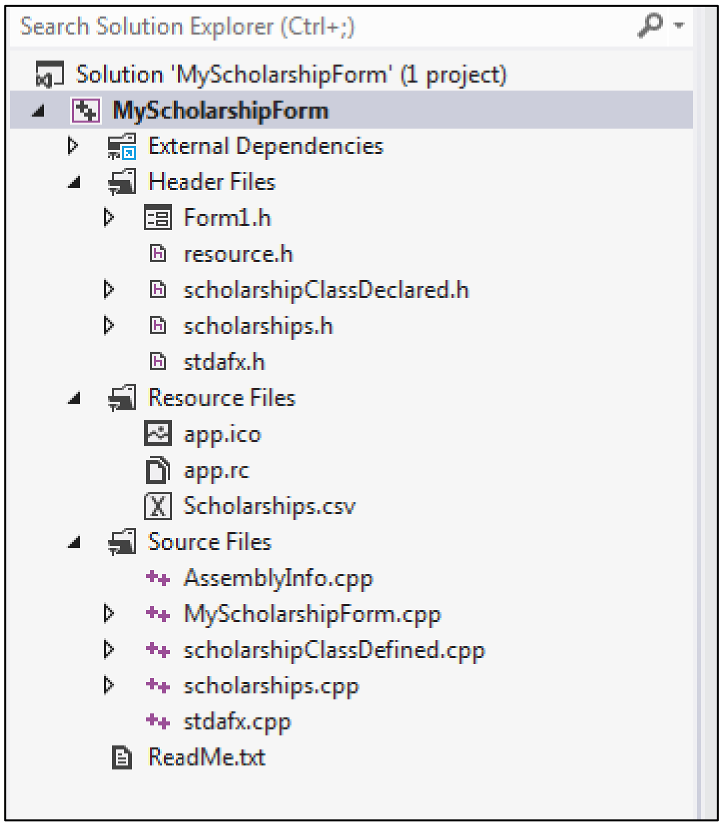
More About Classes Tutorial Part 2 – MyScolarshipForm

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.

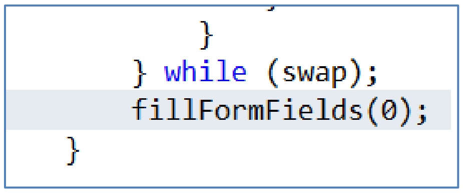


4. **Run** the program to make sure it is set up correctly.

Updating the Form when Necessary

5. **Click** the "*Sort by Last Name*" button and notice that nothing immediately happens.
6. **Click** the **Next** button and then the **Previous** button and notice you are NOT on the original first record. The reason you didn't see a change to the form is because it was not "updated" until you pressed the Next buttons.

7. Add the following line of code to the *btnSortByLastName_Click()* method to "update" the form after the sort button is clicked. Add it right after the do-while loop as highlighted below. This updates the form after the sort and sets focus to the first element in the array, which is now different.

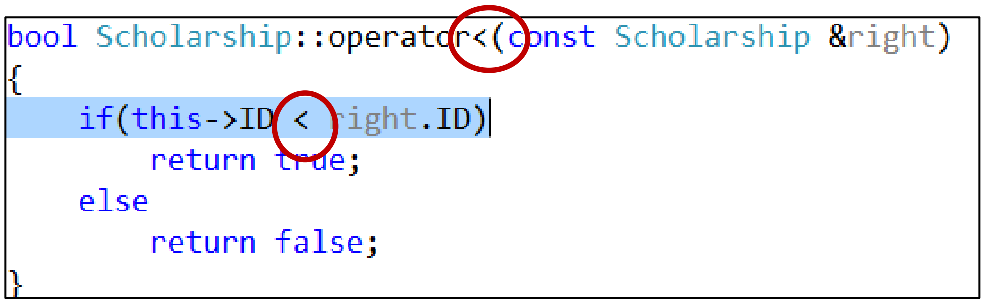


```
            }
        } while (swap);
        fillFormFields(0);
    }
```

8. **Run** the program and see what happens now when you **click** the "*Sort by Last Name*" button. Remember to always reload the form after changes to the location in the array to show the current record.

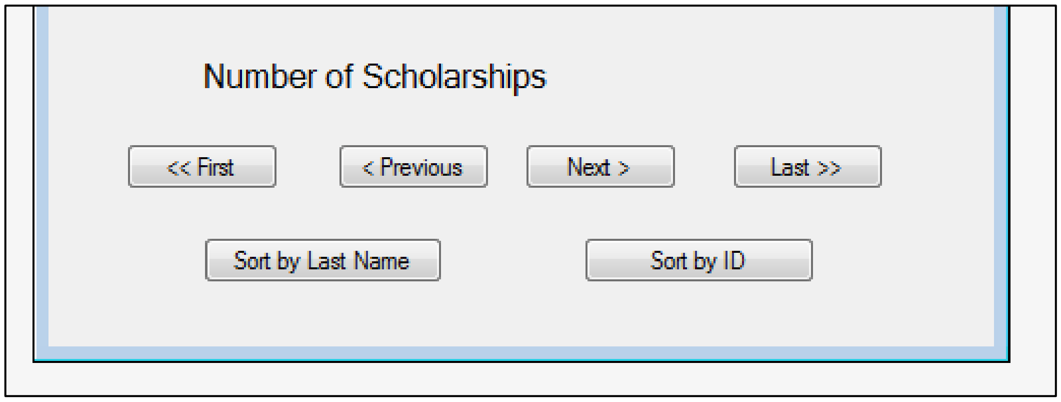
Getting the Order back to ID order

9. **Copy** the ***operator >*** method in the *Scholarship* class and paste it just below the current one. Then change the > symbols to < in the new method and replace the ***Lname*** references to *ID* as shown below.

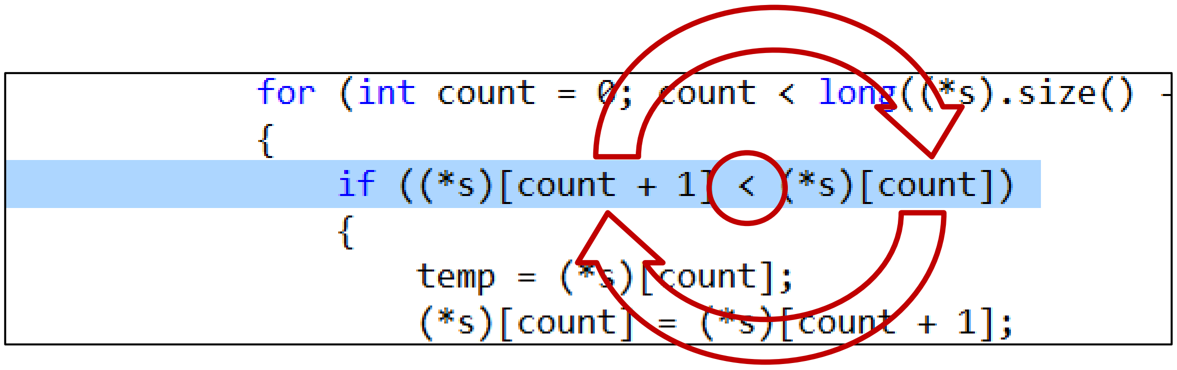


```
bool Scholarship::operator<(const Scholarship &right)
{
    if(this->ID < right.ID)
        return true;
    else
        return false;
}
```

10. **Open** *Form1* in design view and add another button at the bottom. Name the button ***btnSortByID*** and change the text to "Sort by ID." Your form should look similar to the picture below.



11. **Double-click** the new button, **copy** the sort code inside ***btnSortByLname()*** into the new method, and then change the one line of code as shown below. Note how the entire comparison of elements is swapped so that the program will sort in ascending order.

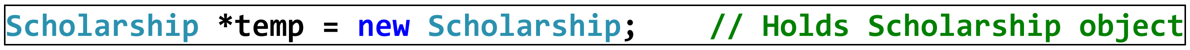


```
        for (int count = 0; count < long((*s).size() -
        {
            if ((*s)[count + 1] < (*s)[count])
            {
                temp = (*s)[count];
                (*s)[count] = (*s)[count + 1];
```

12. **Run** the program and test the new button and method. Debug the program if you find typing errors.

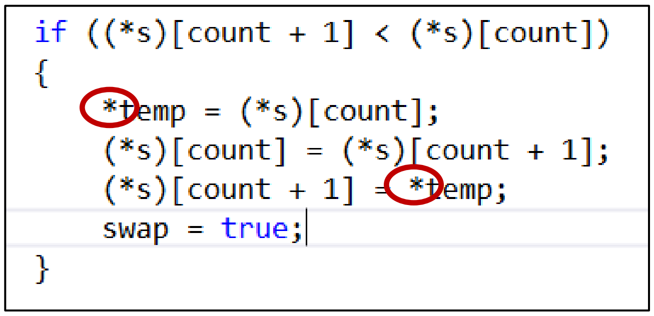
Notice that when you click one of the sort buttons that the Number of Scholarships increases to 100. It changes back to 99 as soon as you click the First, Previous, Next, or Last button. This is because you used a temporary local variable in each sort routine that does NOT "go out of scope" (gets deleted and calls the temporary Scholarship object's destructor) until AFTER the call to *fillFormFields()*. Therefore, the form is refreshed with the current incremented value of the static member *numScholarships* before it is decremented. To fix this, you can change it to a pointer and manually *delete* it just before the call to *fillFormFields().*

13. Change the declaration of the temporary Scholarship object in each sort click method to:



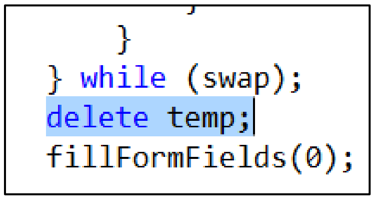
```
Scholarship *temp = new Scholarship;    // Holds Scholarship object
```

14. Because you change the local variable to a pointer, you will need to add the indirection operator (*) to the 2 references to *temp* in the swap block of code as follows.



```
if ((*s)[count + 1] < (*s)[count])
{
    *temp = (*s)[count];
    (*s)[count] = (*s)[count + 1];
    (*s)[count + 1] = *temp;
    swap = true;
}
```

15. Now, **add** the *delete* statement just before the call to *fillFormFields()* as shown below.

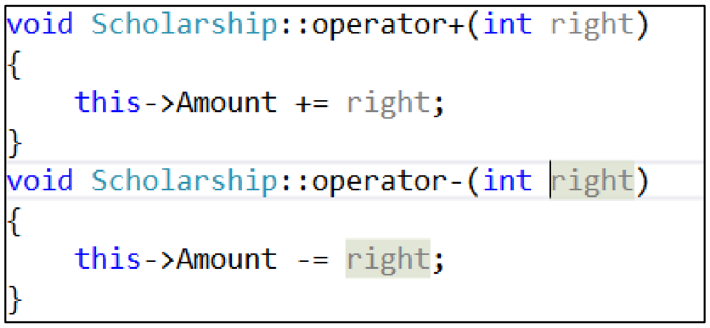


```
        }
    } while (swap);
    delete temp;
    fillFormFields(0);
```

16. **Run** the program and test the buttons to see if the number of Scholarships remains at 99.

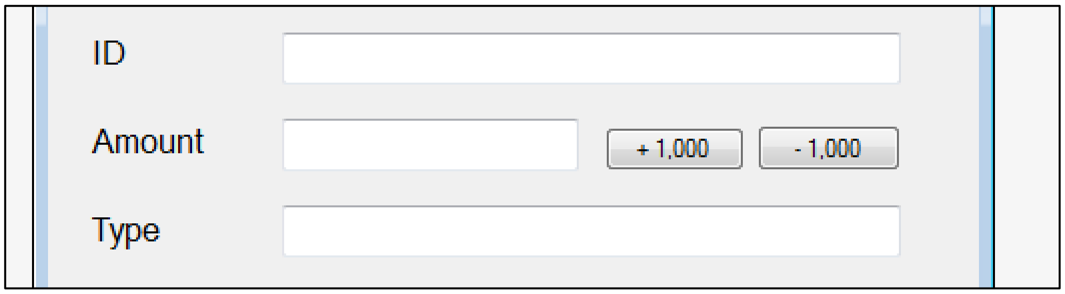
More Overloaded Operators

17. Add the following 2 overloaded operator methods to the *Scholarship* class so that you can easily add and subtract external amounts from the internal “Amount” member variable. You will use this later.



```
void Scholarship::operator+(int right)
{
    this->Amount += right;
}
void Scholarship::operator-(int right)
{
    this->Amount -= right;
}
```

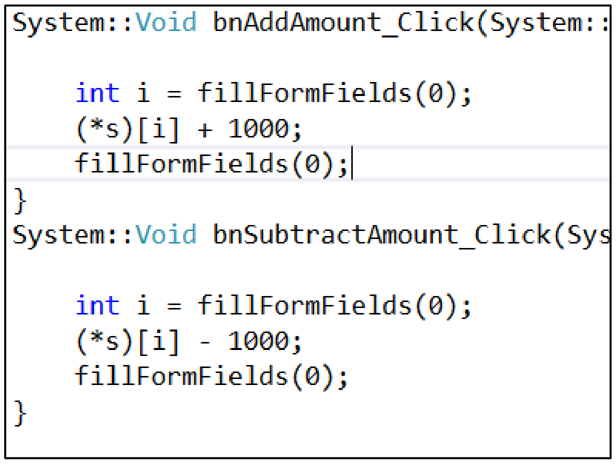
18. **Rebuild** the program to check if you typed the code in correctly.
19. **Resize** the ***Amount*** textbox to make it shorter and add 2 new buttons named ***bnAddAmount*** and ***bnSubtractAmount*** as shown below. Change the text properties for the buttons to ***+ 1,000*** and ***– 1,000***, and then create the methods for them by double-clicking each button.



How these new buttons will work is rather complex. You could simply add to the value that is displayed in the Amount TextBox. However, this does not change the value in the array element unless you write code that copies the new value into the array element. Then, there is the problem of identifying which element to copy the new value to. Though the *fillFormFields()* method contains a static variable that keeps track of what “index” of the array is being used, there is no way to access that number without calling the *fillFormFields()* method, which does not do much good since it is only used to fill the form. You can modify the *fillFormFields()* method a bit to add functionality for the two new buttons. Basically, you can change the “void” return to an “int” so that you can call *fillFormFields(0)* at any time to get a copy of the index of the currently displayed *Scholarship* array element, use it, then call *fillFormFields(0)* again to refresh the From with the new values.

20. Change “**void** fillFormFields(int p)” to “**int** fillFormFields(int p)” and add a “**return i;**” statement at the bottom of the method.

21. Type the following code into the 2 new button methods for adding and subtracting 1,000 to the currently displayed record.



```
System::Void bnAddAmount_Click(System::

    int i = fillFormFields(0);
    (*s)[i] + 1000;
    fillFormFields(0);
}
System::Void bnSubtractAmount_Click(Sys

    int i = fillFormFields(0);
    (*s)[i] - 1000;
    fillFormFields(0);
}
```

22. Run the program and text the new button methods to make sure they work properly.
23. Want to add commas to the From's display of the Amount? You already have the *addCommas*() function in the scholarships.cpp file, so **modify** the line of code in the *fillFormFields()* method of *Form1.h* that updates the Amount's text box as show below. Test the program to see if it adds a comma to the Amount field of the form.

```
    textBoxAmount->Text = gcnew String(addCommas((*s)[i].getAmount()).c_str());
```

    Note that the above code only affects the text in the textbox and does NOT add a comma to the array, itself. This is very crucial in saving the value back to the input file, since you do not want the comma there. This would create problems in reading the file since it is in a Comma Separated Values format.

24. With the program running, add 1,000 to the S-03 scholarship, click the Next button, and then click the Previous button to see that the change is persistent in the array. Note the new value.
25. Close program, then reopen it and look at scholarship S-03. Did the value keep the new value?

Saving Edited Records

26. Run the program, change the last name of the first record, click Next, then Previous. Did the edit save?

This is because you only changed the value in the TextBox and did NOT commit the change to the underlying array that holds all the data. The *add* and *subtract* buttons changed the array first, then refreshed the Form, which is opposite of what you just tried to did in the previous step. You can fix this by creating an "updateData" method that will update the record as soon as you move away from record using one of the navigation buttons.

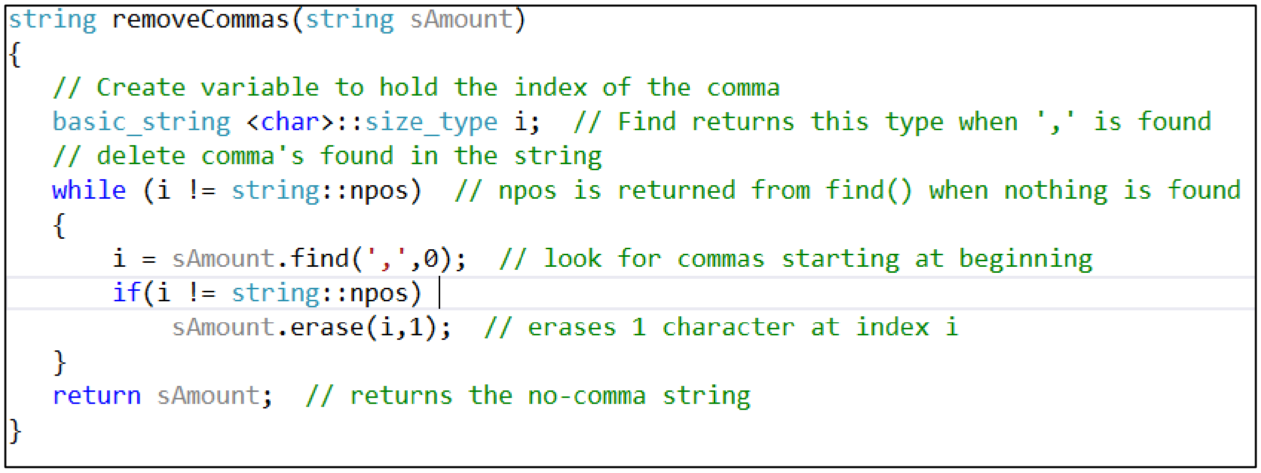
27. **Add** the following method to the *Form1* class. Make sure you add it below the closing ';' of the *bnNew_Click*() method and the "};" that closes the *Form1* class.

```
void updateData()
{
   // First, set up the tool set in Visual C++ that will convert
   //  a System::String^ to a char* that points to an array of characters
   using namespace Runtime::InteropServices;
   // Second, Collect all the textBox values and convert to a char*
   //  this must be done before the call to fillFormFields()
   //  otherwise, the textboxes are refreshed before getting the new data
   char *pID = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxID->Text).ToPointer());
   char *pType = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxType->Text).ToPointer());
   char *pLength = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxLength->Text).ToPointer());
   char *pDate = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxStarts->Text).ToPointer());
   char *pLname = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxLastName->Text).ToPointer());
   char *pFname = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxFirstName->Text).ToPointer());

   // get index of current record
   int i = fillFormFields(0);   // This wipes out the data you just entered

   // Update the current record in the array of objects
   (*s)[i].setID(pID);
   (*s)[i].setType(pType);
   (*s)[i].setLength(pLength);
   (*s)[i].setDate(pDate);
   (*s)[i].setLname(pLname);
   (*s)[i].setFname(pFname);
}
```

28. **Add** a call to the new method at the beginning of EVERY button click method on the form (except for the Add and Subtract buttons).
29. Run the program and test out editing the fields and moving around with various buttons. Then go back to edited records to see if they saved properly.

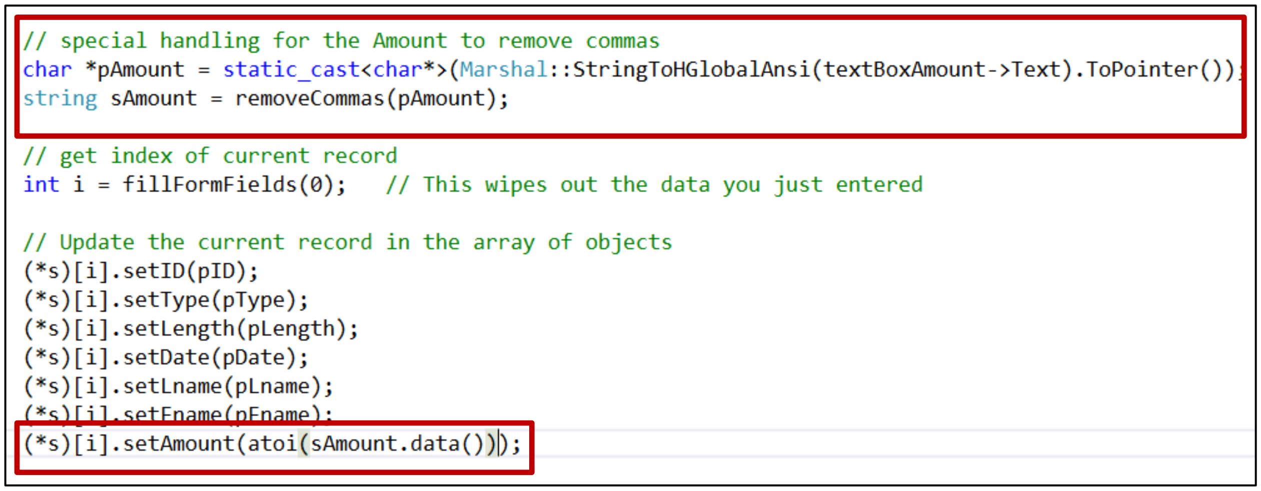
You will notice that the Amount field was not saved if you edited it. This is because the field in the Textbox is a string with a comma in it that needs to be removed before saving the value back to the array record. The next step will show you how to remove commas from strings so they can be converted to integers for storage.

30. **Add** the following function to the *Scholarships.cpp* file to remove commas from any string passed to it.



```
string removeCommas(string sAmount)
{
   // Create variable to hold the index of the comma
   basic_string <char>::size_type i;  // Find returns this type when ',' is found
   // delete comma's found in the string
   while (i != string::npos)  // npos is returned from find() when nothing is found
   {
       i = sAmount.find(',',0);  // look for commas starting at beginning
       if(i != string::npos) 
           sAmount.erase(i,1);  // erases 1 character at index i
   }
   return sAmount;  // returns the no-comma string
}
```

31. Add this special handling code to the *updateData()* method of Form1 to save the changed Amount input.



```
// special handling for the Amount to remove commas
char *pAmount = static_cast<char*>(Marshal::StringToHGlobalAnsi(textBoxAmount->Text).ToPointer());
string sAmount = removeCommas(pAmount);

// get index of current record
int i = fillFormFields(0);   // This wipes out the data you just entered

// Update the current record in the array of objects
(*s)[i].setID(pID);
(*s)[i].setType(pType);
(*s)[i].setLength(pLength);
(*s)[i].setDate(pDate);
(*s)[i].setLname(pLname);
(*s)[i].setFname(pFname);
(*s)[i].setAmount(atoi(sAmount.data()));
```

32. **Run** and test your program. Debug it if necessary.

You have 2 *writeFile()* functions in the *scholarships.cpp* file. However, these just right reports for printing and do not save updated records back to the original input file. Therefore, you need to add a new function to this file to save the edited data.

33. **Create** a new function in the *scholarships.cpp* file as follows. You place this function here so that if you ever reuse the Scholarship class you can copy this file with it as an external helping function, since it has nothing to do with the *Form*, but with the array of Scholarship objects. Note how the new function uses a *pointer* to the array to write the output.

```
bool saveFile(vector<Scholarship> *s) // s is a pointer to the array
{
    // Make a back up of the current input file before changes
    system("copy Scholarships.csv, Scholarships.bak");

    // Open the input file to be overwritten with new data
    ofstream fout;
    fout.open("Scholarships.csv");
    if(!fout)
        return false;  // Output flie failed to open. Data not saved.

    // Overwrite the existing input file
    int sArraySize = s->size();          // Get the size of the array
    for(int i = 0; i < sArraySize; i++)
    {
        fout << (*s)[i].getID() << ','
             << (*s)[i].getAmount() << ','
             << (*s)[i].getType() << ','
             << (*s)[i].getLength() << ','
             << (*s)[i].getDate() << ','
             << (*s)[i].getLname() << ','
             << (*s)[i].getFname();
        // Do NOT add a line feel to the very last record
        //  to avoid errors while reading the file in the next time
        if(i != sArraySize - 1)
            fout << endl;  // does not add the end of line if at the end
    }
    return true;  // Overwrite successful
}
```

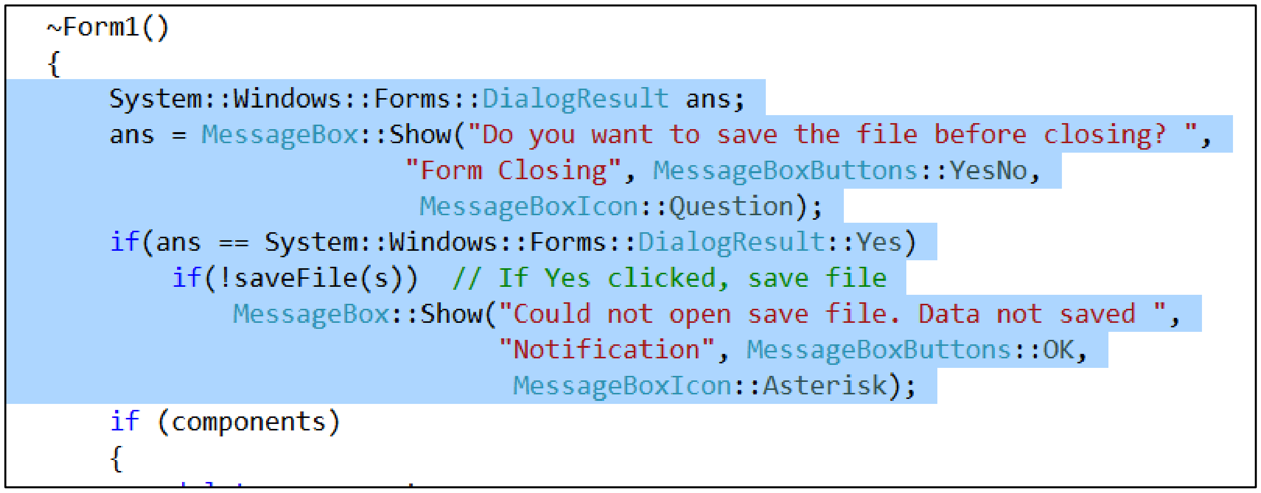
34. **Add** a new button to the *Form* and name it ***bnSave***. Create a method for it and use the following code for the new method.

```
private: System::Void bnSave_Click(System::Object^  sender, System::EventArgs^  e) {

             if(!saveFile(s))  // Pass by address to the function
                    MessageBox::Show("Could not open save file. Data not saved ",
                        "Notification", MessageBoxButtons::OK,MessageBoxIcon::Asterisk);
         }
```

35. **Run** the program and test the new button. **Debug** your code if necessary.
36. With the program running, **sort** the data by *Last Name* and **Save** the file. **Close** the program and look at the ***Scholarships.csv*** file to see if it saved in the new order properly. Look to make sure the last line feed did NOT get added to the new file. **Open** the newly created *Scholarships.bak* file to make sure the backup file was created.
37. **Run** the program again. **Sort** by ID and save the file. Then close it and look at the newly saved file again.

Now that you can save the file, any updates you make will be saved when you click the button, but what about if the user simply closes the program? The file will only save right now if the user clicks the *Save* button. You can add a method that will ask the user if the form is to be saved or not by placing more code in the form's Destructor.

38. **Add** the following highlighted code to the form's Destructor method.

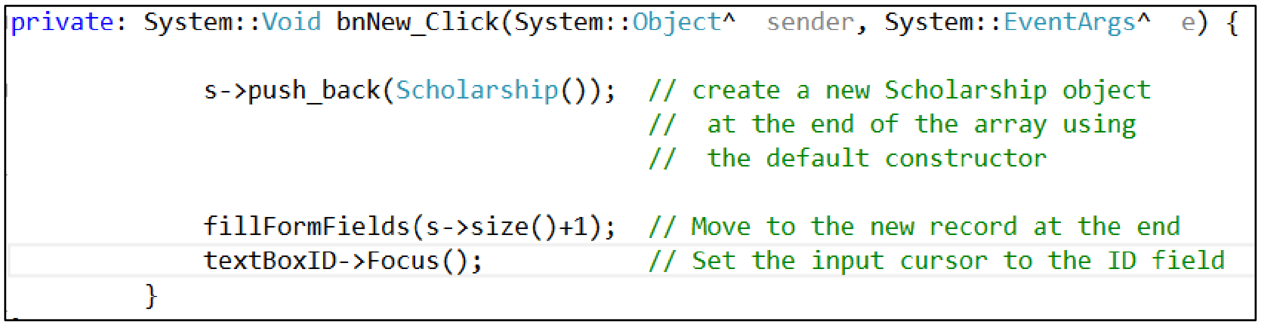


```
~Form1()
{
    System::Windows::Forms::DialogResult ans;
    ans = MessageBox::Show("Do you want to save the file before closing? ",
                           "Form Closing", MessageBoxButtons::YesNo,
                            MessageBoxIcon::Question);
    if(ans == System::Windows::Forms::DialogResult::Yes)
        if(!saveFile(s))  // If Yes clicked, save file
            MessageBox::Show("Could not open save file. Data not saved ",
                             "Notification", MessageBoxButtons::OK,
                              MessageBoxIcon::Asterisk);
    if (components)
    {
```

Adding New Records

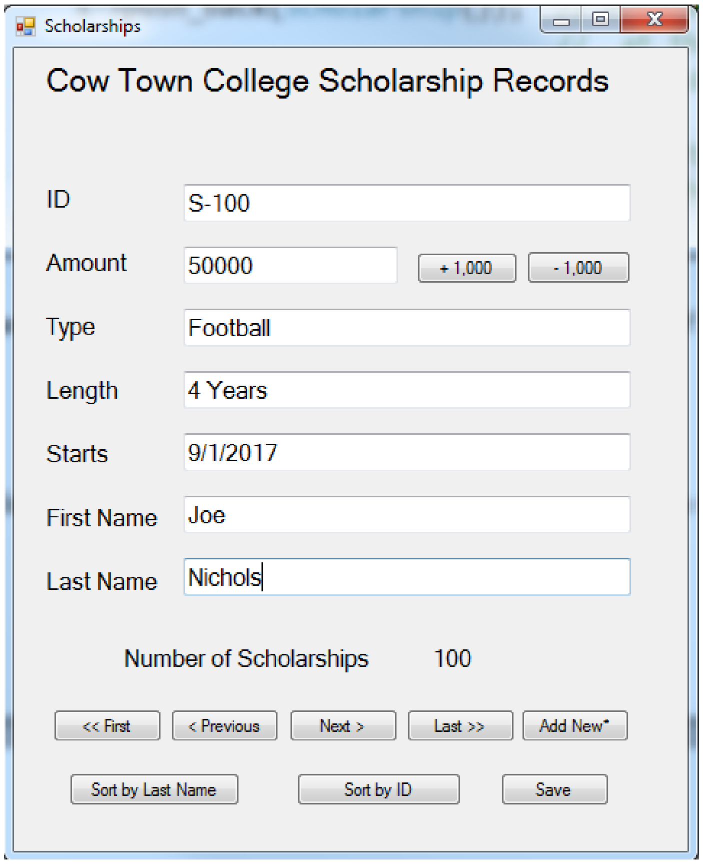
It is simple now to add new records to the file. First you create a blank record at the end of the array and fill in the data. When the file is saved the new record is added to the file.

39. Add a new button to the form and name it ***bnNew***. Change it's text property to "**Add New***". Create a method for the new button and use the following code for it.



```
private: System::Void bnNew_Click(System::Object^  sender, System::EventArgs^  e) {

             s->push_back(Scholarship());  // create a new Scholarship object
                                           //  at the end of the array using
                                           //  the default constructor

             fillFormFields(s->size()+1);  // Move to the new record at the end
             textBoxID->Focus();           // Set the input cursor to the ID field
         }
```

40. Run the program and add the following record.



Scholarships

Cow Town College Scholarship Records

ID	S-100
Amount	50000
Type	Football
Length	4 Years
Starts	9/1/2017
First Name	Joe
Last Name	Nichols

Number of Scholarships 100

Note the Number of Scholarships changes from 99 to 100.

41. **Click** the **Previous** and **Next** buttons to see that it saved to the array of objects.

42. **Save** the file, **close** the program, **run** the program again to see if the record is still there with 100 records now showing.