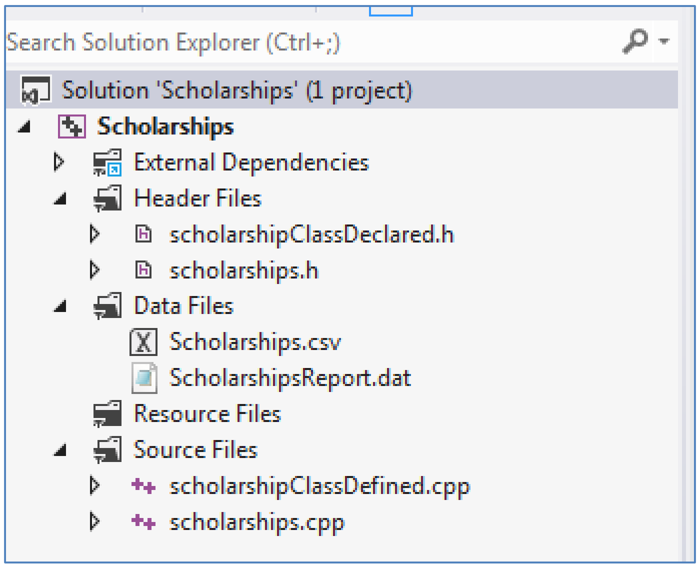
Vector Arrays Tutorial Part 2 - Scholarships

This tutorial continues the demonstration of how to use vector arrays to read a Comma Separated Value (CSV) file and output simple reports to both the console and a file. It uses the Scholarship class to create a vector array of objects when read in from the file. 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.

1. Create an empty Visual Studio ***Win32 Console Application*** on the **DESKTOP** named ***Scholarships***.
2. Download the Part 2 tutorial ZIP file from Blackboard>(This Lesson) and save to your computer.
3. Using Windows Explorer, COPY all of the of the files from the ZIP file into the new project's folder located at **…Desktop\Scholarship\Scholarship**
4. With the new project open in *Visual Studio*, add the files to the different filters of the ***Solution Explorer*** panel so that your project looks like the following screen shot. You will need to add the ***Data Files*** filter to add the scholarships.csv file to the project in the correct location as shown.



5. **Run** the program to make sure it works properly. It should only display an empty console output with "Press any key to continue…" as output.
6. Open the ***scholarships.cpp*** file and read the description at the top to find out more about the program and what the data file's structure is.
7. Open the ***scholarshipClassDeclared.h*** file and read through it to get familiar with its structure. It is a simple class with 2 constructors and all the sets and gets written, but no error checking.
8. Open the ***Scholarships.csv*** file to get familiar, once again, with its structure. Notice each of its lines have the "fields" separated by commas. This makes it easier to "tokenize" into separate variables after reading each line into a string.

9. First, you will use the knowledge you have about the use of *setw()* and *iomanip* alignments to edit the code in *writeFile*() and properly place the fields where they belong below the headings. You will also use your knowledge of arrays to do a little formatting of a few of the fields. Currently, the report output looks like this (which is obviously not lined up correctly):

```
****************************Scholarships Report****************************
**************************************************************************

ID      Amount      Type      Length      Starts     First Name     Last Name
--------------------------------------------------------------------------
S-01  40000  Football2 years  1/29/2016  Joel  Barthmaier
```

10. *Iomanip* is already included in the *Scholarships.h* file, so go to the *writeFile*() function and modify the code as shown below to line up the fields underneath the headings properly. Note the use of the "   " spaces after the output of right aligned fields so that the next left aligned field is in the proper location.

```cpp
void writeFile(Scholarship s,     // Pass in by value- no need to change string in i
               ofstream &fout)   // Also pass in the output file buffer by ref to w
{
    createReportHeadings(fout);
    fout << s.getID() <<  right << setw(10) << s.getAmount()
         << "    " << left <<  setw(12) << s.getType()
         << setw(10) << s.getLength() << right << setw(10) << s.getDate()
         << "   " << left <<  setw(15) << s.getFname() << s.getLname() << endl;
}
```

11. Run the program and your new report should look like this (modify your code from the previous step if it does not):

```
****************************Scholarships Report****************************
**************************************************************************

ID      Amount      Type      Length      Starts     First Name     Last Name
--------------------------------------------------------------------------
S-01     40000    Football    2 years     1/29/2016   Joel           Barthmaier
```

12. Now, add a comma to the amount for output. Add the following function to the end of the *Scholarships.cpp* file (don't forget to add the prototype to the *Scholarships.h* file). Study the code to learn how it works.

```cpp
string addCommas(int num)  // Adds commas to any number for formatted output to report files
{
   string s = to_string(num);  // convert the number to string to hold the formatted number
   // insert comma's from right (at implied decimal point) to left
   int sSize = s.size(); // index to last digit
   if (sSize > 3)
      for (int i = (sSize - 3);  i > 0;  i -= 3)
         s.insert(i,",");  // look up the string's insert() method to see how it works
                           //  it is found on page 816 of the textbook or using "help"
   return s;
}
```

13. Add a call to the new function from inside the *writeFile*() function so that the comma is displayed in the output. Use the code in bold below.

```
fout << s.getID() << right << setw(10) << addCommas(s.getAmount())
```

14. Run the program to make sure it compiles correctly, making corrections to your typing as necessary. You should see the comma appear in the output file. Note that the new function does NOT change the original integer number inside the object.
15. Your *Scholarship* class has no error checking to prevent bad data from getting into the class' properties. Modify all the "*set*" methods in the *ScholarshipClassDefined.cpp* file as shown in the following code.

```
bool Scholarship::setID(string id)
{
    if( id.empty())
    {
        ID = "invalid";
        return false;
    }
    ID = id;
    return true;
}
bool Scholarship::setAmount(int a)
{
    if( a < 0)
    {
        Amount = 0;
        return false;
    }
    Amount = a;
    return true;
}
```

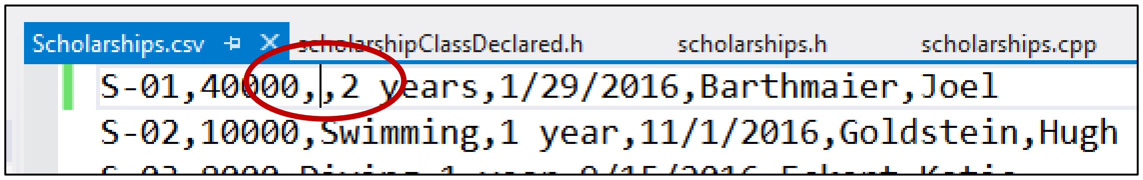
```
bool Scholarship::setType(string t)
{
    if( t.empty())
    {
        Type = "invalid";
        return false;
    }
    Type = t;
    return true;
}
bool Scholarship::setLength(string len)
{
    if( len.empty())
    {
        Length = "invalid";
        return false;
    }
    Length = len;
    return true;
}
```

```
bool Scholarship::setDate(string d)
{
    if( d.empty())
    {
        DateStarts = "invalid";
        return false;
    }
    DateStarts = d;
    return true;
}
bool Scholarship::setLname(string ln)
{
    if( ln.empty())
    {
        Lname = "invalid";
        return false;
    }
    Lname = ln;
    return true;
}
```

```
bool Scholarship::setFname(string fn)
{
    if( fn.empty())
    {
        Fname = "invalid";
        return false;
    }
    Fname = fn;
    return true;
}
```

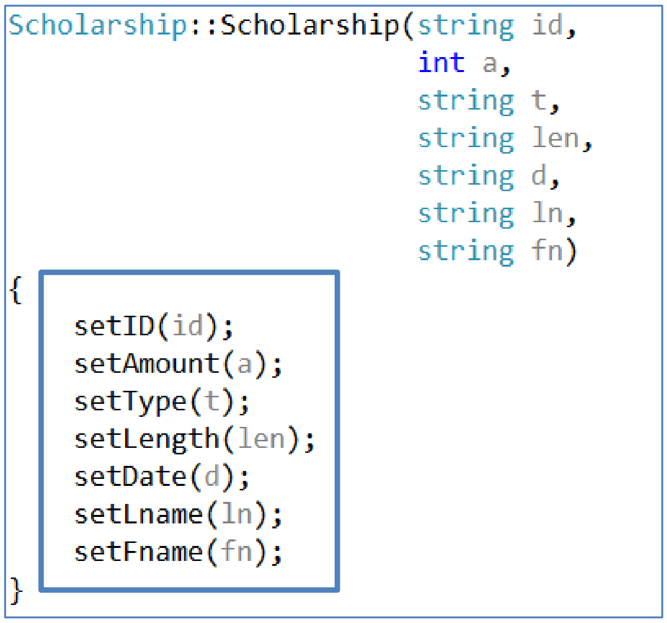
Note that you are changing all the "set" methods to return a type *bool*, so change the prototypes as well.

16. **Rebuild** the program to make sure you entered everything correctly. If you get errors, go back and correct your typing.
17. Open the *Scholarships.cvs* file and delete the word "Football" from the first record, but leave the commas as shown below. Save the file.



```
S-01,40000,,2 years,1/29/2016,Barthmaier,Joel
S-02,10000,Swimming,1 year,11/1/2016,Goldstein,Hugh
```

18. **Run** the program and look at the output report. You will notice that the word "*invalid*" does not appear, but just an empty place where "*Football*" should be. This is because the file is being read in using the overloaded constructor and NOT the "*set*" methods of the class which contain the error checking.
19. **Rewrite** the overloaded constructor in the *Scholarship* class to use the "*set*" methods that you just modified as shown below.

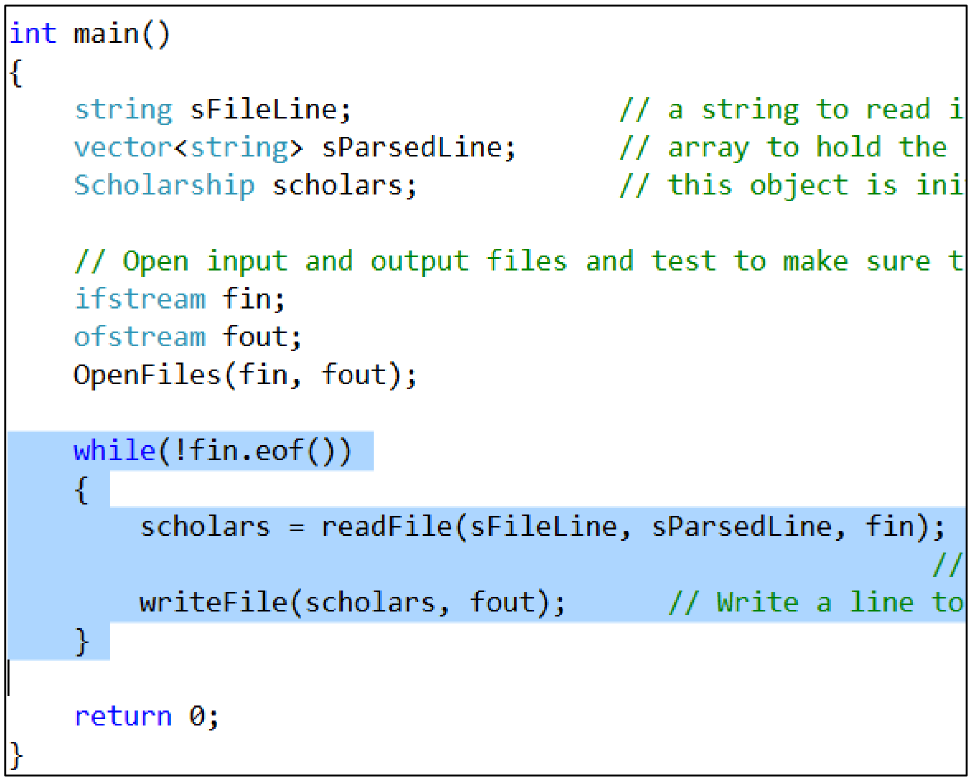


```
Scholarship::Scholarship(string id,
                         int a,
                         string t,
                         string len,
                         string d,
                         string ln,
                         string fn)
{
    setID(id);
    setAmount(a);
    setType(t);
    setLength(len);
    setDate(d);
    setLname(ln);
    setFname(fn);
}
```

20. **Run** the program again to see what happens to the report file.
21. Open the *Scholarships.cvs* file and add "*Football*" back in its correct location between the commas and save the file. **Run** the program once more to make sure you correctly added the field back to the file.

Now that you have beefed up your class a little, you are ready to work on the main program that uses the class to loop through and create the entire report.

22. Modify your *main()* function so that the Read and Write statements are in a loop that does not end until the end of the input file is reached. Your modified *main()* should look the code below.

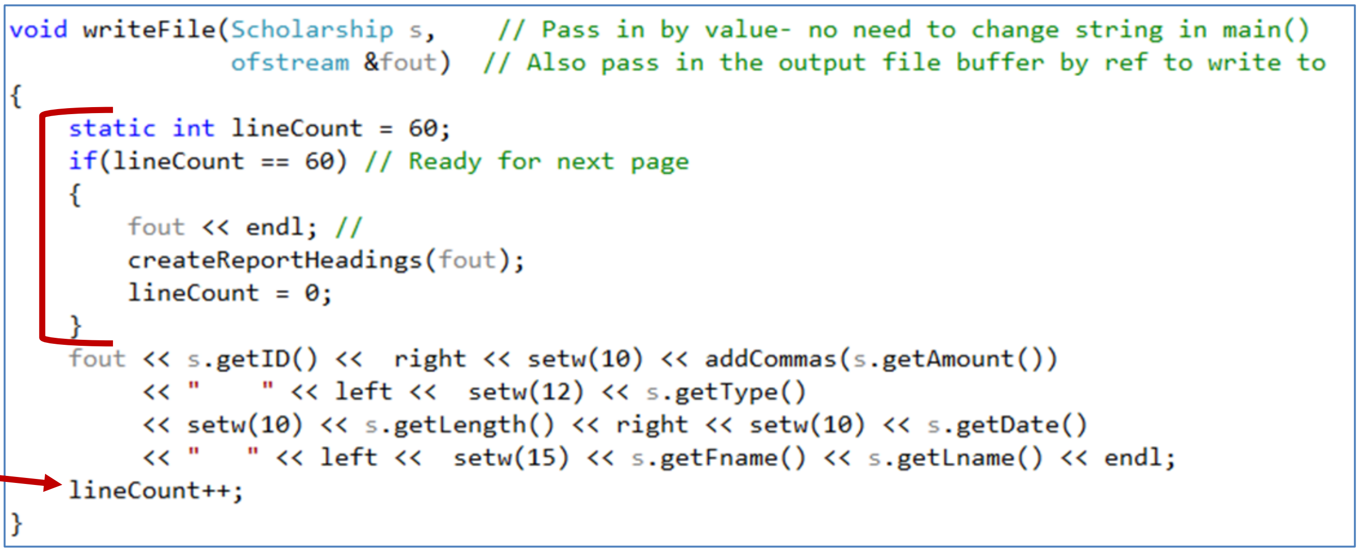


```
int main()
{
    string sFileLine;                // a string to read i
    vector<string> sParsedLine;      // array to hold the
    Scholarship scholars;            // this object is ini

    // Open input and output files and test to make sure t
    ifstream fin;
    ofstream fout;
    OpenFiles(fin, fout);

    while(!fin.eof())
    {
        scholars = readFile(sFileLine, sParsedLine, fin);
                                                         //
        writeFile(scholars, fout);      // Write a line to
    }

    return 0;
}
```

23. **Run** the program and look at the new output file....what a mess you just made of it. LOL.

You need to modify some code to get it looking good and only print the headings once per page. Let's say you only want 60 records to print per page, which is about what a normal printer would print for this report before ejecting the page and starting a new one. You do this by adding a static line counter to the *writeFile*() function and placing the call to *createReportHeadings*() inside an *if* statement. Then, increment the counter each time a record is printed. The counter is reset each time the headings are printed.

24. **Modify** the *writeFile*() function with the code as indicated in the picture below.

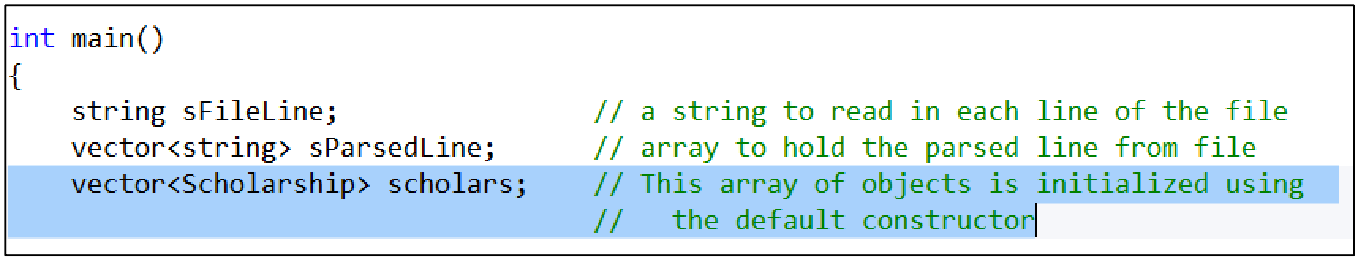


```
void writeFile(Scholarship s,    // Pass in by value- no need to change string in main()
               ofstream &fout)  // Also pass in the output file buffer by ref to write to
{
    static int lineCount = 60;
    if(lineCount == 60) // Ready for next page
    {
        fout << endl; //
        createReportHeadings(fout);
        lineCount = 0;
    }
    fout << s.getID() <<  right << setw(10) << addCommas(s.getAmount())
         << "     " << left <<  setw(12) << s.getType()
         << setw(10) << s.getLength() << right << setw(10) << s.getDate()
         << "    " << left <<  setw(15) << s.getFname() << s.getLname() << endl;
    lineCount++;
}
```

25. **Run** the program and look at the new output file. Scroll down the report file and see that the headers reprinted after every 60 records. While the file is open in Visual Studio, select **File > Print** to see if it truly ejects a page at the right location. If the printer does not eject at the right place, adjust the ***lineCount*** setting up or down so that the headings reprint at the top of each printer page.

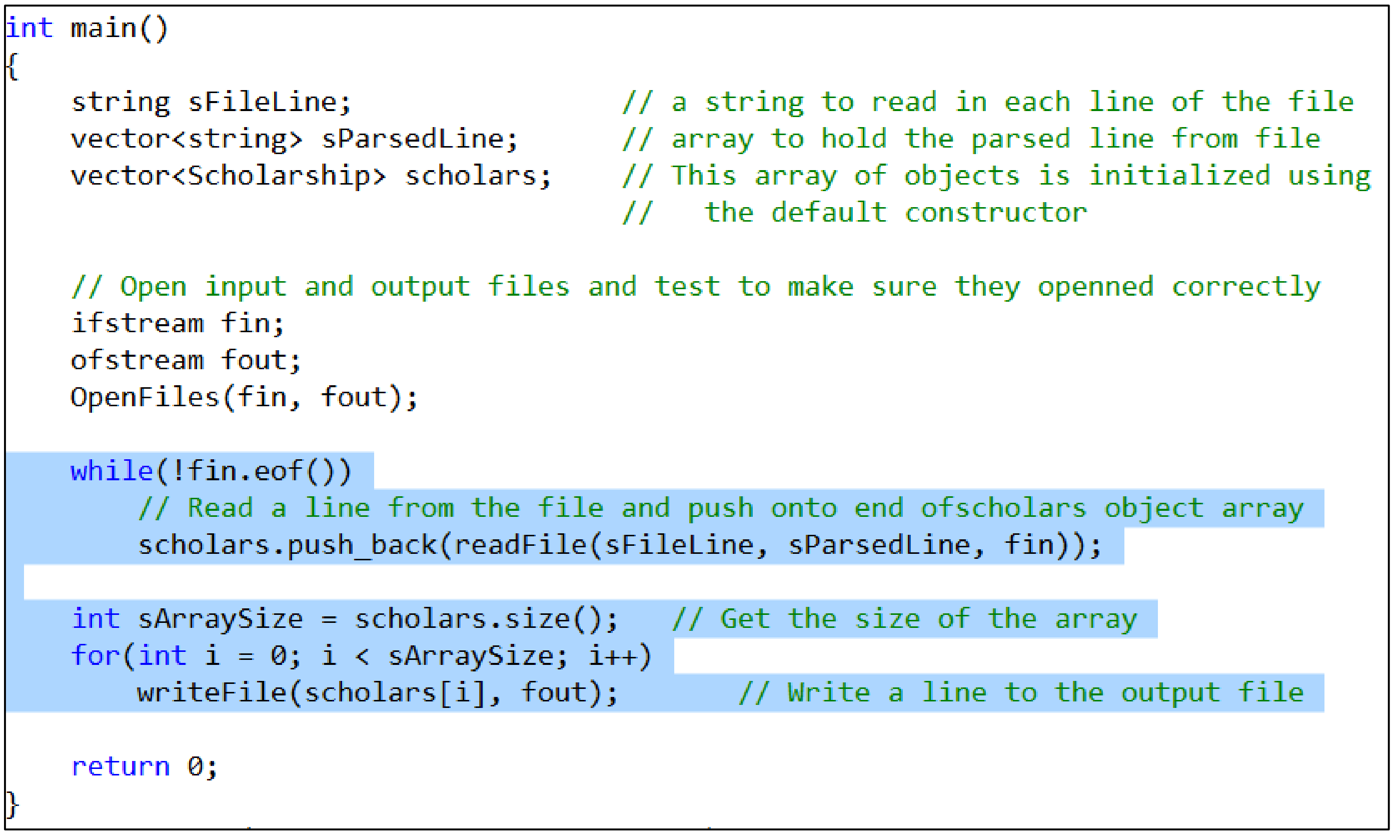
Now you are ready to do some high-powered programming by reading the entire file into a vector array of objects then output the report all at once or even re-write the input file after editing some of the records.

26. Change the single ***Scholarship*** object created at the top of *main*() to a *vector* array as seen in the highlighted code below. Also update the comment to reflect change.



```
int main()
{
    string sFileLine;               // a string to read in each line of the file
    vector<string> sParsedLine;     // array to hold the parsed line from file
    vector<Scholarship> scholars;   // This array of objects is initialized using
                                    //   the default constructor
```

27. Now, **modify** the input and output statements in *main*() so that it uses one loop to collect the file into the array of objects and a second loop to write them to the report file. Your new changes to *main*() should look like the highlighted code below. Be sure to type the comments to keep track of what you are doing in the program. You use the vector's "*push_back*" method as you did in the parsing routine in the last tutorial to add the newly returned Scholar object returned from the *readFile*() function to the end of the array. You also use the vector's "*size*" method to find out how many records are in the array. You use an index "`[i]`" to pass one object at a time from the array to the *writeFile*() function.



```
int main()
{
    string sFileLine;               // a string to read in each line of the file
    vector<string> sParsedLine;     // array to hold the parsed line from file
    vector<Scholarship> scholars;   // This array of objects is initialized using
                                    //   the default constructor

    // Open input and output files and test to make sure they openned correctly
    ifstream fin;
    ofstream fout;
    OpenFiles(fin, fout);

    while(!fin.eof())
        // Read a line from the file and push onto end ofscholars object array
        scholars.push_back(readFile(sFileLine, sParsedLine, fin));

    int sArraySize = scholars.size();   // Get the size of the array
    for(int i = 0; i < sArraySize; i++)
        writeFile(scholars[i], fout);       // Write a line to the output file

    return 0;
}
```

28. **Run** the program and inspect the output report file to make sure it still looks the same. Debug and correct any errors that you may encounter at this time before moving on.

With the entire file now copied into your array of objects, you can now do many things with the data before sending it to the output file. Start by collecting data for a summary report showing the number of scholarships and the total amount of all scholarship money.

29. Add the following function (to include the comments) to the end of the *Scholarships.cpp* file to write the summary report. As always, don't forget to add the prototype to the header file.

```
void createReportSummary(vector<Scholarship> sArray,  // Pass by value (copy) the entire array
                         ofstream &fout)             // Pass the output file by reference
{
    int total = 0;                              // Accumulator for total amount
    int sArraySize = sArray.size();             // Get the size of the array

    // Loop through the array to accumulate the total amount of all scholarships
    for(int i = 0; i < sArraySize; i++)
        total += sArray[i].getAmount();         // Add the Amount of each scholarship to the total

    // Write the summary report output line
    fout << "\n\n" << "Summary Report \n"
         << "        Total Number of Scholarships: "
         << right << setw(12) << sArraySize << endl
         << "        Total Scholarship Amount:    $ "
         << setw(11) << addCommas(total);
}
```

30. Now call the new function after the report has been run by adding the following highlighted code and comments to the bottom of *main*() just before the "*return 0;*" line.

```
int main()
{
    . . .
    int sArraySize = scholars.size();       // Get the size of the array
    for(int i = 0; i < sArraySize; i++)
        writeFile(scholars[i], fout);       // Write a line to the output file

    // Write the summary report to the end of the outut file by calling the
    //  summary function and passing both the array of objects and the
    //  file handle to the function
    createReportSummary(scholars, fout);

    return 0;
}
```

31. **Run** the program, correct any errors you made and view the output report file to see the summary report at the end of the report. If the alignments are off, go back and correct your mistakes in the summary report function.

32. **Modify** the *createReportSummary()* function to “beef up” the summary report some by collecting the amounts for each type of scholarship in the array. The code below shows the changes you make to this function. To be ***kind***, you can open the ***Step 32 – createReportSummary.txt*** file provided in this tutorial’s zip folder, then ***copy and paste*** this entire function into your program to REPLACE the previous version of *createSummaryReport().* Just be sure to study the code to see what is happening here.

```
void createReportSummary(vector<Scholarship> sArray,  // Pass by value (copy) the entire array
                         ofstream &fout)              // Pass the output file by reference
{
    int total = 0;                          // Accumulator for total amount
    int sArraySize = sArray.size();         // Get the size of the array
    // Accumulators for type totals
    int baseTotal = 0, baskTotal = 0, dTotal = 0, fTotal = 0, gTotal = 0, softTotal = 0,
        swimTotal = 0, tTotal = 0, vTotal = 0;

    // Loop through the array to accumulate the total amount of all scholarships
    for(int i = 0; i < sArraySize; i++)
    {
        total += sArray[i].getAmount();    // Add the Amount of each scholarship to the total
        string sType = sArray[i].getType(); // Get the type of scholarship for this record
        // Add the Type's Amount to the appropriate accumulator
        if(sType == "Baseball") baseTotal += sArray[i].getAmount();
        else if(sType == "Basketball") baskTotal += sArray[i].getAmount();
        else if(sType == "Diving") dTotal += sArray[i].getAmount();
        else if(sType == "Football") fTotal += sArray[i].getAmount();
        else if(sType == "Golf") gTotal += sArray[i].getAmount();
        else if(sType == "Softball") softTotal += sArray[i].getAmount();
        else if(sType == "Swimming") swimTotal += sArray[i].getAmount();
        else if(sType == "Track") tTotal += sArray[i].getAmount();
        else if(sType == "Volleyball") vTotal += sArray[i].getAmount();
    }
    // Write the summary report output line
    fout << "\n\nSummary Report \n"
         <<     "--------------\n"
         << "        Total Number of Scholarships: "
         << right << setw(12) << sArraySize << endl
         << "        Baseball:                    $ "
         << setw(11) << addCommas(baseTotal) << endl
         << "        Basketball:                    "
         << setw(11) << addCommas(baskTotal) << endl
         << "        Diving:                        "
         << setw(11) << addCommas(dTotal) << endl
         << "        Football:                      "
         << setw(11) << addCommas(fTotal) << endl
         << "        Golf:                          "
         << setw(11) << addCommas(gTotal) << endl
         << "        Softball:                      "
         << setw(11) << addCommas(softTotal) << endl
         << "        Swimming:                      "
         << setw(11) << addCommas(swimTotal) << endl
         << "        Track:                         "
         << setw(11) << addCommas(tTotal) << endl
         << "        Volleyball:                    "
         << setw(11) << addCommas(vTotal) << endl
         << "        ===========================================\n"
         << "        Total Scholarship Amount:    $ "
         << setw(11) << addCommas(total);
}
```

33. **Run** the program and view the new summary report at the end of the output report file.

What if the client wants to run reports for each type of scholarship? You can do this by overloading the *writeFile*() function to include a string filter parameter that will write only the scholarship type report requested.

34. Add the following overloaded *writeFile*() function to the bottom of the *Scholarships.cpp* file. You can do this **easily** by copying the existing *writeFile()* function, pasting it to the bottom of the file, and adding the code that is highlighted below.

```
    // Overloaded function for report of specific type of scholarship
    void writeFile(Scholarship s,    // Pass in by value- no need to change string in main()
                           ofstream &fout,  // Also pass in the output file buffer by ref to write to
                           string sType)    // Pass in by value the type of scholarship
    {
        static int lineCount = 60;
        if(lineCount == 60) // Ready for next page
        {
                fout << endl; //
                createReportHeadings(fout);
                lineCount = 0;
        }
        if(sType == s.getType())
        {
                fout << s.getID() <<  right << setw(10) << addCommas(s.getAmount())
                         << "    " << left <<  setw(12) << s.getType()
                         << setw(10) << s.getLength() << right << setw(10) << s.getDate()
                         << "   " << left <<  setw(15) << s.getFname() << s.getLname() << endl;
                lineCount++;
        }
    }
```

Surely you added the prototype to the *Scholarship.h* file.

35. Now, back in ***main***(), add the following highlighted code between the call to ***createReportSummary()*** and the ***return 0;*** lines.

```
    //  file handle to the function
    createReportSummary(scholars, fout);

    // Close the original output report to reuse its file handler
    //  and not overwrite the original output report
    fout.close();
    fout.open("ScholarshipsReportByType.dat");
    if(!fout)
    {
        cout << "Output file did not open. Program will exit." << endl;
        exit(0);
    }
    // Ask the user for the specific report
    string sType;
    cout << "What specific type of scholarship would you like a report on?\n";
    cin  >> sType;
    cout << endl;
    for(int i = 0; i < sArraySize; i++)
        writeFile(scholars[i], fout, sType); // Write a line to the output file

    return 0;
}
```

36. **Run the program and correct any errors that may occur during compiling.**
37. **Add** the newly created ***ScholarshipsReportByType.dat*** file to the ***Data Files*** filter of the VS project. Then open the file to make sure it recorded correctly. It should show only the type that the user entered. Run the program several more times using different types of scholarships as input to see that it is working with all types.
38. Close Visual Studio.
39. Be sure to save your project to your USB drive for future reference.
40. If you are in the college computer lab, delete the entire project folder from the Desktop.