LCOV - code coverage report

			Hit	Total	C
Current view:	top level - Users/raksha/Downloads/expenses-cpp-repo-test/new/src - newexpenses.cpp (source / functions)				
Test:	coverage.info	**Lines:**	73	80	
Date:	2022-02-10 09:41:21	**Functions:**	7	7	1

```
    Line data    Source code
  1             : /* This program takes csv as input file
  2             :  * containing monthly expenses per category line item and
  3             :  * calculates the average monthly cost of an expense per category.
  4             :  * Also calculates the average monthly overall cost based on entire year's total expenses.
  5             :  * Using csv parser library, rapidcsv.h from https://github.com/d99kris/rapidcsv
  6             :  */
  7             : #include <iostream>
  8             : #include <fstream>
  9             : #include <iomanip>
 10             : #include <vector>
 11             : #include <cmath>
 12             : #include <string>
 13             : #include <algorithm>
 14             : #include "../include/rapidcsv.h"
 15             :
 16             : using namespace std;
 17             : using namespace rapidcsv;
 18             :
 19             : /*This constant is for limiting length of Columns evaluated for each month of the year. */
 20             : const int MAX_SIZE = 12;
 21             :
 22          22 : string usage = "Usage: make run filename=<filename>";
 23          22 : string details = "Please correct and rerun program. Only enter numbers without any symbols or characters for each expense amount.\nE
 24             :
 25             : /* The getDoc fcn will open csv file using rapidcsv::Document class
 26             :  * This Document class uses input stream ifstream to open and read csv,
 27             :  * and will throw an iostream exception if the stream has failbit or badbit set to true.
 28             :  * Note that it is set it to skip empty lines and accept / char as beginning of line as comment prefix
 29             :  * to skip comment lines.
 30             :  */
 31             : Document getDoc(char* filename)
 32             : {
 33             :     try
 34             :     {
 35          27 :         Document doc( filename,
 36             :         /* Using Label parameters to label 1st column and 1st row without comments as headers*/
 37           9 :         LabelParams(0, 0),
 38             :         /* Using default delimiter comma */
 39           9 :         SeparatorParams(),
 40             :         /*Using default converter parameters, which will not convert invalid numbers and will throw exception by default*/
 41           9 :         ConverterParams(),
 42             :         /*LineReader Params set to true to skip comment lines starting with '/', and also set to skip empty lines*/
 43           9 :         LineReaderParams(true, '/', true) );
 44             :
 45           7 :         return doc;
 46          11 :     }
 47             :
 48             :     /*Exception handling for incorrect file name, file type, or missing filename as input parameter. */
 49             :     catch(const ios_base::failure& io_err)
 50             :     {
 51          12 :         cerr << "Unable to open file: " << filename <<". Please try again using correct filename and path."<< endl<<usage<<endl;
 52           4 :         cerr <<" Input/Output error reading file: "<< io_err.what();
 53           0 :         exit(1);
 54           0 :     }
 55             :
 56           9 : }
 57             :
 58             : /* This fcn will Get Column header names and check values to determine if input file is valid csv with required headers.
 59             : */
 60             : vector<string> getColHeader(Document doc)
 61             : {
 62             :     /*Creating vector of strings for month values to validate with column header vector*/
 63         189 :     vector<string> months ={"jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"};
 64           7 :     vector<string> col= doc.GetColumnNames();
 65             :
 66           7 :     if(col.empty())
 67             :     {
 68           3 :         cerr<<"ERROR - this is not a valid csv file. Please enter correct filename. "<<usage<<endl;
 69           0 :         exit(1);
 70             :     }
 71             :
 72             :     /* checking to make sure no additional column headers were added beyond the 12 months */
 73           6 :     if (col.size()>MAX_SIZE)
 74             :     {
 75             :         /*csv should actually contain 13 columns, but rapidcsv lib does not count first column since it is considered part of row he
 76          10 :         cerr<<"Column headers out of range --parsing data only within 13 total columns. Please keep format intact where 1st column i
 77           5 :         col.resize(MAX_SIZE);
 78           5 :     }
 79             :
 80             :     /* Change formatting of months in csv header to lowercase and 3-ltr abbreviation to validate */
 81             :
 82          12 :     if(col != months)
 83         130 :     {   for(unsigned i=0; i< col.size(); i++)
 84             :         {
 85         120 :             if(col.at(i).size() > 3)
 86             :                 {
 87          10 :                     col.at(i).resize(3);
```

```
88          5 :                 }
89        300 :             transform(col.at(i).begin(), col.at(i).end(), col.at(i).begin(), ::tolower);
90            :             /*debug
91            :             cout<<col.at(i)<<endl;
92            :             */
93         60 :         }
94          5 :     }
95            :
96         12 :     if(col != months)
97            :     {
98          0 :         cerr<<"ERROR - this file is not the correct csv! Please enter csv containing expenses for each month. "<<usage<<endl;
99          0 :         exit(1);
100           :     }
101           :
102         6 :     return col;
103        12 : }
104           :
105           : /* This getRow fn will create vector the row header names to get the expense categories.
106           :  */
107           :
108           : vector<string> getRowHeader(Document doc)
109           : {
110        12 :             vector<string> row = doc.GetRowNames();
111         6 :             return row;
112        12 : }
113           :
114           : /* The setAve fcn will determine average of each expense category.
115           : *  Average is not necessarily based on entire year, but rather on number of values entered in given category.
116           : *  This is to account for values entered mid-year, or if expenses did not start at beginning of year, or end at year-end.
117           : *  Also calculates average total expense per month.
118           : */
119           : void setAve( Document doc, vector<string> col, vector<string> row)
120           : {
121        12 : float sum =0 ,ave=0, totalAve=0;
122           : try
123           : {
124       108 :     for (unsigned i=0; i< row.size();i++)
125           :     {
126           :         /*create vector for each row of values per expense category line item */
127       106 :         vector<float> cost =doc.GetRow<float>(row.at(i));
128        48 :         if(cost.size()>MAX_SIZE)
129           :         {  /* instead of throwing custom exception msg to end program, sending error msg and continuing program to output remaining
130           :             *   throw out_of_range("out_of_range");
131           :             */
132        25 :             cerr<<"Out of Range: Need to remove extra items in row:"<<row.at(i)<<". Resizing row to only calculate average of values
133         5 :             cost.resize(MAX_SIZE);
134         5 :         }
135      1122 :         for (unsigned j=0; j<cost.size();j++)
136           :         {
137      1026 :             if (!isnan(cost.at(j))|| !isinf(cost.at(j)))
138           :             {
139      1026 :                 sum += cost.at(j);
140       513 :             }
141           :             else
142           :             {
143         0 :                 throw invalid_argument("Invalid computation");
144           :
145           :             }
146       513 :         }
147           :         /* Average calcualated based on number of values enterd in given row just in case expenses did not last for duration of enti
148           :          * denominator will always be non-zero since getRow fcn does not input any empty rows
149           :          */
150        48 :         ave = sum/cost.size();
151           :
152        48 :         if(!isnan(ave)|| !isinf(ave))
153           :         {
154       432 :             cout<<"Average monthly expense for "<< row.at(i) << " is $" << fixed<< setprecision(2)<< ave << endl;
155           :
156        48 :         }
157           :         else
158           :         {
159         0 :            throw invalid_argument("Invalid computation");
160           :         }
161           :         /*aggregate each line item average to get average monthly total for all expenses*/
162        48 :         totalAve += ave;
163           :         /*reset sum for next row iteration in loop*/
164        48 :         sum = 0;
165        48 :     }
166         4 :     cout << "Total monthly average cost for all expenses is $ " << totalAve  << "." << endl;
167           :
168         8 : }//end try
169           :
170           : catch(const invalid_argument& inv_err)
171           : {
172        28 :     cerr<<"Error: "<< inv_err.what()<<".  Unable to convert to float value -Incorrect data entered in csv file!"<<details+usage<<enc
173           :
174         4 : }
175           :
176           : catch(const out_of_range& e)
177           : {
178         6 :     cerr<<"Invalid value exceeding range of float value in csv! Error:"<<e.what() << details+ usage<< endl;
179         5 : }
180           :
181           :
182        11 : }
183           : int main(int argc,char* argv[])
184           : {
185           :     try
```

```
186            :     {
187         11 :         if (argc==2)
188            :         {
189          7 :             char* filename = argv[1];
190          7 :             Document csv = getDoc(filename);
191         19 :             vector<string> col = getColHeader(csv);
192         18 :             vector<string> row = getRowHeader(csv);
193         30 :             setAve(csv,col,row);
194          6 :          }
195            :         else
196            :         {
197          4 :             throw invalid_argument("Incorrect command line arguments specified");
198            :         }
199         10 :     }
200            :     catch (const invalid_argument& e)
201            :     {
202         10 :         cout << "Error: " << e.what() << endl<<usage<<endl;
203          2 :     }
204          8 :     return 0;
205          2 : }
```

Generated by: LCOV version 1.15-5-g462f71d