Julius Ranoa

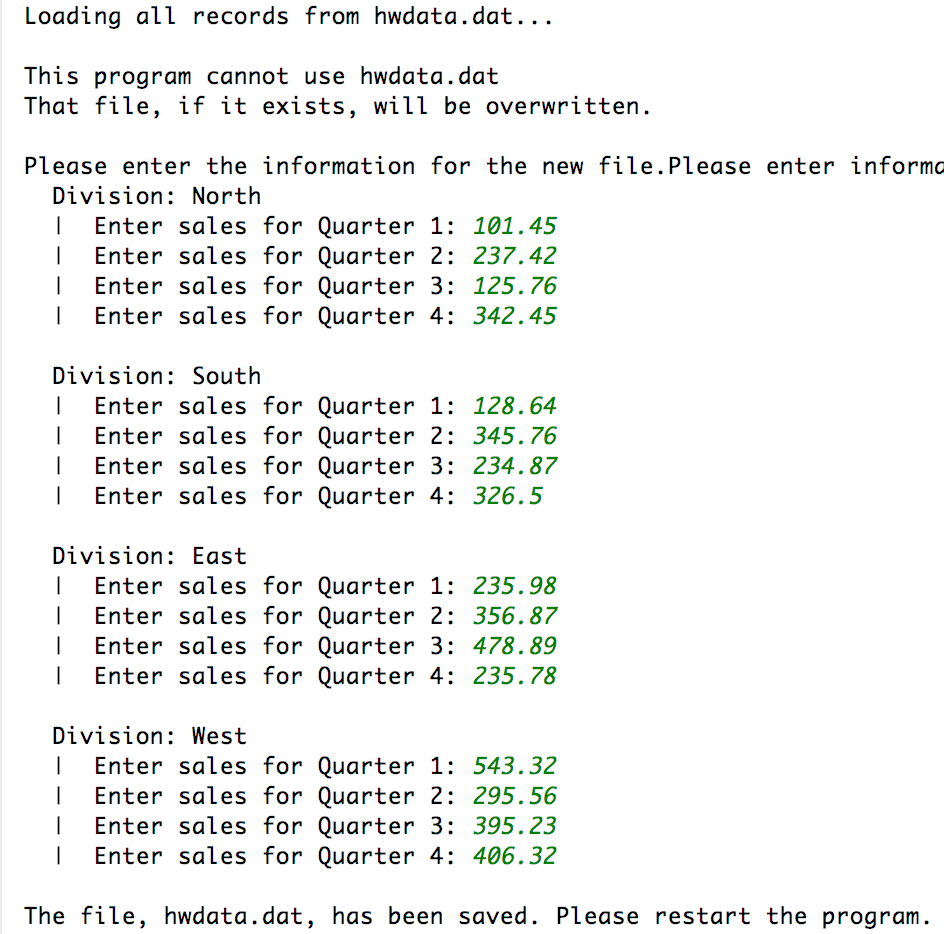
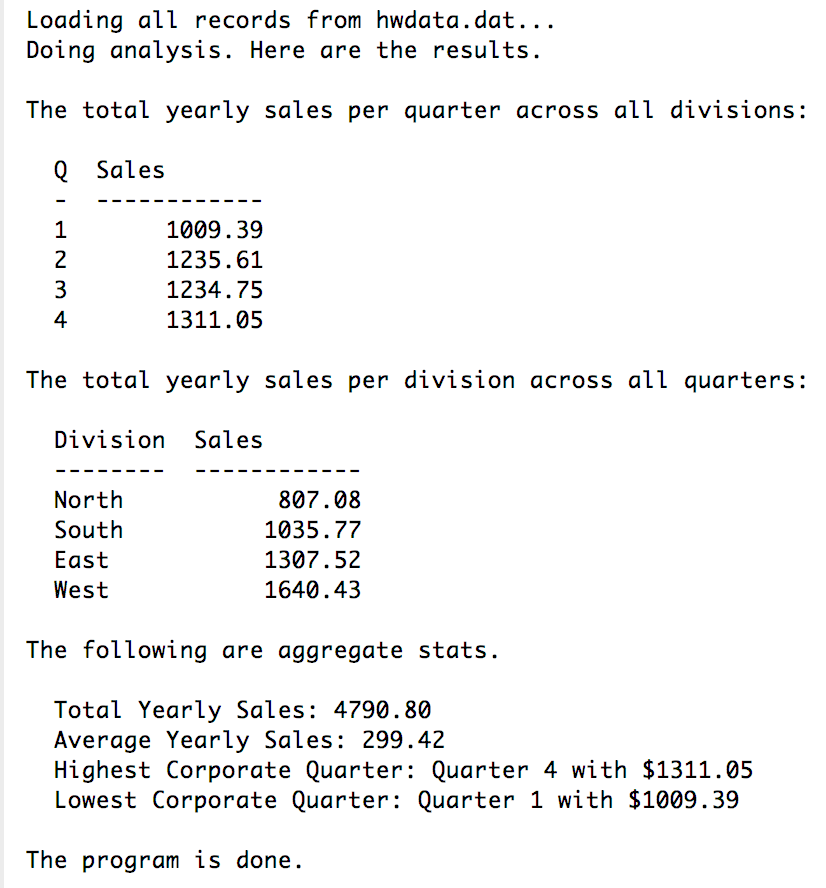
CSC 121 001 Computer Science I

Homework

Chapter 13 Advanced File and I/O Operations

Programming Challenge Qn. 13 and 14 Corporate Sales Data I/O.

Screenshots.

1. When *source* provided doesn’t refer to an existing file.  
   
2. When *source* provided refers to an existing file.  
   

Source Code

**main.cpp**

#include **"Driver.h"**#include **<iostream>**#include **<iomanip>  
using namespace** std;  
  
**int** main() {  
 Driver d;  
 string source = **"hwdata.dat"**;  
  
 cout << **"Loading all records from "** << source << **"..."** << endl;  
 d.deserializeAll(source);  
 **if** (!d.good()) {  
 cout << endl;  
 cout << **"This program cannot use "** << source << endl;  
 cout << **"That file, if it exists, will be overwritten. "** << endl << endl;  
 cout << **"Please enter the information for the new file."**;  
 d.getInput();  
 d.serializeAll(source);  
 cout << **"The file, "** << source << **", has been saved. Please restart the program."**;  
 **return** -1;  
 }  
  
 d.doCalculations();  
 cout << **"Doing analysis. Here are the results."** << endl << endl;  
  
 cout << **"The total yearly sales per quarter across all divisions: "** << endl << endl;  
 cout << **" Q Sales "** << endl;  
 cout << **" - ------------"** << endl;  
 cout << fixed << showpoint;  
 **for** (numValuePair nv : d.getTotalQuarterlySales()) {  
 cout << **" "**;  
 cout << setw(1) << setprecision(0) << nv.num << **" "**;  
 cout << setw(12) << setprecision(2) << right << nv.value << endl;  
 }  
 cout << endl;  
  
 cout << **"The total yearly sales per division across all quarters: "** << endl << endl;  
 cout << **" Division Sales "** << endl;  
 cout << **" -------- ------------"** << endl;  
 cout << fixed << showpoint;  
 **for** (nameValuePair nv : d.getTotalDivisionSales()) {  
 cout << **" "**;  
 cout << setw(8) << left << nv.name << **" "**;  
 cout << setw(12) << setprecision(2) << right << nv.value << endl;  
 }  
 cout << endl;  
  
 cout << **"The following are aggregate stats. "** << endl << endl;  
 cout << **" Total Yearly Sales: "** << setprecision(2) << d.getTotalSales() << endl;  
 cout << **" Average Yearly Sales: "** << setprecision(2) << d.getAverageSales() << endl;  
 cout << **" Highest Corporate Quarter: "** << **"Quarter "** << d.getHighestCorporateQuarter().num  
 << **" with $"** << d.getHighestCorporateQuarter().value << endl;  
 cout << **" Lowest Corporate Quarter: "** << **"Quarter "** << d.getLowestCorporateQuarter().num  
 << **" with $"** << d.getLowestCorporateQuarter().value << endl;  
 cout << endl;  
  
 cout << **"The program is done."** << endl;  
 *// cout << "For reference, here are all the records saved in " << source << endl << endl;  
 // d.printAll();* **return** 0;  
}

**CorporateSales.h**#ifndef **HOMEWORK\_CH13\_QN13\_14\_CORPORATESALES\_H**#define **HOMEWORK\_CH13\_QN13\_14\_CORPORATESALES\_H**#include **<string>**#include **<fstream>  
  
class** CorporateSales {  
  
**private**:  
 std::string divisionName;  
 **int** quarter;  
 **double** sales;  
  
**public**:  
 CorporateSales() {  
 divisionName = **"North"**;  
 quarter = 1;  
 sales = 0.0;  
 }  
 CorporateSales(std::string d, **int** q, **double** s) {  
 divisionName = d;  
 quarter = q;  
 sales = s;  
 }  
  
 *// Getters* std::string getName() **const** { **return** divisionName; }  
 **int** getQuarter() **const** { **return** quarter; }  
 **double** getSales() **const** { **return** sales; }  
  
 *// Setters* **void** setName(std::string n) { divisionName = n; }  
 **void** setQuarter(**int** q) { quarter = q; }  
 **void** setSales(**double** s) { sales = s; }  
  
 **void** serialize(std::ofstream&);  
 **void** deserialize(std::ifstream&);  
 **void** print();  
};  
  
  
#endif *//HOMEWORK\_CH13\_QN13\_14\_CORPORATESALES\_H*

**CorporateSales.cpp**

#include **"CorporateSales.h"**#include **<iostream>  
using namespace** std;  
  
**void** CorporateSales::serialize(ofstream &outFile) {  
 *// Order of Serialization: Quarter > Sales > Name* outFile.write(**reinterpret\_cast**<**char** \*>(&quarter), **sizeof**(quarter));  
 outFile.write(**reinterpret\_cast**<**char** \*>(&sales), **sizeof**(sales));  
  
 **int** l = divisionName.length();  
 outFile.write(**reinterpret\_cast**<**char** \*>(&l), **sizeof**(l));  
 outFile.write(divisionName.data(), l);  
}  
  
  
**void** CorporateSales::deserialize(ifstream &inFile) {  
 **const int** BUFFER\_SIZE = 256;  
 **static char** buffer[256];  
  
 inFile.read(**reinterpret\_cast**<**char** \*>(&quarter), **sizeof**(quarter));  
 inFile.read(**reinterpret\_cast**<**char** \*>(&sales), **sizeof**(sales));  
  
 **int** l = 0;  
 inFile.read(**reinterpret\_cast**<**char** \*>(&l), **sizeof**(l));  
 inFile.read(buffer, l);  
 buffer[l] = **'\0'**;  
  
 divisionName = buffer;  
}

**Driver.h**#ifndef **HOMEWORK\_CH13\_QN13\_14\_DRIVER\_H**#define **HOMEWORK\_CH13\_QN13\_14\_DRIVER\_H**#include **"CorporateSales.h"**#include **<vector>  
  
struct** numValuePair {  
 **int** num;  
 **double** value;  
};  
  
**struct** nameValuePair {  
 std::string name;  
 **double** value;  
};  
  
**class** Driver {  
  
**private**:  
 **bool** isGood = **false**;  
 std::vector<CorporateSales> records;  
 std::vector<numValuePair> totalQuarterlySales;  
 std::vector<nameValuePair> totalDivisionSales;  
 **double** totalSales;  
 **double** averageSales;  
 numValuePair lowestCQuarter;  
 numValuePair highestCQuarter;  
  
**public**:  
 **void** serializeAll(std::string filename);  
 **void** deserializeAll(std::string filename);  
 **bool** good() { **return** isGood; }  
 **void** printAll();  
  
 **void** randomize();  
  
 **void** getInput();  
 **void** doCalculations();  
 **void** calcTotalQuarterlySales();  
 **void** calcTotalDivisionSales();  
 **void** calcTotalnAverageSales();  
 **void** findExtrema();  
  
 std::vector<CorporateSales> getRecords() { **return** records; };  
 std::vector<numValuePair> getTotalQuarterlySales() {  
 **return** totalQuarterlySales;  
 };  
 std::vector<nameValuePair> getTotalDivisionSales() {  
 **return** totalDivisionSales;  
 };  
 **double** getTotalSales() { **return** totalSales; };  
 **double** getAverageSales() { **return** averageSales; };  
 numValuePair getLowestCorporateQuarter() { **return** lowestCQuarter; };  
 numValuePair getHighestCorporateQuarter() { **return** highestCQuarter; };  
};  
  
  
#endif *//HOMEWORK\_CH13\_QN13\_14\_DRIVER\_H*

**Driver.cpp**

#include **"Driver.h"**#include **<iostream>**#include **<iomanip>  
using namespace** std;  
  
**void** Driver::serializeAll(std::string filename) {  
 ofstream file(filename, ios::binary);  
 **if** (file) {  
 **for** (CorporateSales cs : records) {  
 cs.serialize(file);  
 }  
 }  
}  
  
**void** Driver::deserializeAll(std::string filename) {  
 ifstream file(filename, ios::binary);  
 **if** (file) {  
 **while**(file.good() && file.peek() != **EOF**) {  
 CorporateSales cs;  
 cs.deserialize(file);  
 records.push\_back(cs);  
 }  
 isGood = **true**;  
 } **else** isGood = **false**;  
}  
  
**void** Driver::getInput() {  
 vector<string> divisions = {  
 **"North"**, **"South"**, **"East"**, **"West"** };  
 vector<**int**> quarters = {  
 1, 2, 3, 4  
 };  
 **double** temp;  
  
 records.clear();  
 cout << **"Please enter information as prompted."** << endl;  
 **for** (string div : divisions) {  
 cout << **" Division: "** << div << endl;  
 **for** (**int** q : quarters) {  
 cout << **" | Enter sales for Quarter "** << q << **": "**;  
 cin >> temp;  
 records.push\_back(CorporateSales(div, q, temp));  
 }  
 cout << endl;  
 }  
}  
  
**void** Driver::printAll() {  
 cout << **" "** << **"Division "** *// 9* << **"Q "** *// 2* << **"Sales "** << endl; *// 10* cout << **" "** << **"-------- "** << **"- "** << **"----------"** << endl;  
  
 cout << fixed << showpoint;  
 **for** (CorporateSales cs : records ) {  
 cout << **" "**;  
 cout << setw(8) << left << cs.getName() << **" "**;  
 cout << setw(1) << cs.getQuarter() << **" "**;  
 cout << setw(10) << setprecision(2) << right << cs.getSales();  
 cout << endl;  
 }  
}  
  
**void** Driver::randomize() {  
 vector<string> divs = { **"North"**, **"South"**, **"East"**, **"West"** };  
 vector<**int**> qs = {1, 2, 3, 4};  
  
 srand(time(**NULL**));  
 **for** (string div : divs) {  
 **for** (**int** q : qs) {  
 records.push\_back(  
 CorporateSales(div, q, rand() % 1000 + 1)  
 );  
 }  
 }  
}  
  
**void** Driver::doCalculations() {  
 calcTotalQuarterlySales();  
 calcTotalDivisionSales();  
 calcTotalnAverageSales();  
 findExtrema();  
}  
  
**void** Driver::calcTotalQuarterlySales() {  
 totalQuarterlySales = {  
 {1, 0}, {2, 0}, {3, 0}, {4, 0}  
 };  
  
 **for** (CorporateSales cs : records) {  
 **switch** (cs.getQuarter()) {  
 **case** 1:  
 totalQuarterlySales[0].value += cs.getSales();  
 **break**;  
 **case** 2:  
 totalQuarterlySales[1].value += cs.getSales();  
 **break**;  
 **case** 3:  
 totalQuarterlySales[2].value += cs.getSales();  
 **break**;  
 **case** 4:  
 totalQuarterlySales[3].value += cs.getSales();  
 **break**;  
 }  
 }  
}  
  
**void** Driver::calcTotalDivisionSales() {  
 totalDivisionSales = {  
 {**"North"**, 0}, {**"South"**, 0}, {**"East"**, 0}, {**"West"**, 0}  
 };  
  
 **for** (CorporateSales cs : records) {  
 **for** (**int** i = 0; i < totalDivisionSales.size(); i++) {  
 **if** (cs.getName().compare(totalDivisionSales[i].name) == 0) {  
 totalDivisionSales[i].value += cs.getSales();  
 **break**;  
 }  
 }  
 }  
}  
  
**void** Driver::calcTotalnAverageSales() {  
 totalSales = 0;  
 **for** (CorporateSales cs : records) {  
 totalSales += cs.getSales();  
 }  
 averageSales = totalSales / records.size();  
}  
  
**void** Driver::findExtrema() {  
 lowestCQuarter = highestCQuarter = totalQuarterlySales[0];  
 **for** (**int** i = 1; i < totalQuarterlySales.size(); i++) {  
 **if** (totalQuarterlySales[i].value < lowestCQuarter.value) {  
 lowestCQuarter = totalQuarterlySales[i];  
 } **else if** (totalQuarterlySales[i].value > highestCQuarter.value) {  
 highestCQuarter = totalQuarterlySales[i];  
 }  
 }  
}