ECS 60 Programming Assignment #4, 50 points (1:50 coding) Spring 2014

Due: Wednesday, May 21st, at 11:59pm to p4 directory
Primary class file names: stats.h, stats.cpp, sales.h, sales.cpp. Executable names: stats.out, sales.out
Makefile names: MakefileStats, MakefileSales (You must handin both Makefiles!)

For this assignment you will do two smaller, challenge programs. The first has to do with keeping baseball statistics, and the second involves keeping track of the sales of a retail store. Remember to put the authors' names on the first line of stats.h, and sales.h.

Baseball Stats (25 points, 50 minutes)

As you may know, I am a San Francisco Giants baseball fan. I often go to their website where Major League Baseball (MLB) maintains sortable lists of player stats based on teams and the league as a whole. These lists are maintained in real time. Your Stat class will maintain these lists for the player's "batting average". A "batting average" is the number of safe hits divided by the total number of "at bats". To keep things simple, we will say that each offensive turn a player has (called an "at bat") he either hits the baseball safely or makes an out. Thus, to keep the system up to date, it must be notified whenever a player makes an out, or has a safe hit. For instance, if a player hits safely in an at bat, then that player's hit and "at bat" totals are automatically incremented instantly, and his batting average will increase (unless he is already batting 1.00).

1. Stat class specifications
   - 1.1. Updates specify a player's name, team, and whether he made a hit, or an out.
     - 1.1.1. Your teams and league should start empty, and grow in size as updates arrive for new players.
   - 1.2. Queries specify the name of a team or "MLB" to receive the names of the players with the top 10 batting averages, sorted by batting averages at that moment, for the specified team, or the league as a whole. The system allows players with identical averages to be in any order.
   - 1.3. There are approximately 100 times more updates than queries. About half of the queries are about the league as a whole.
2. Players specifications
   - 2.1. Names are no longer than 24 chars. Names are guaranteed unique within a team, but there may be duplicates within the league.
   - 2.2. Each player can have no more than 600 at bats. The average player gets a hit about a quarter of the time.
   - 2.3. To accurately compute a player's batting average(which MUST be calculated as a double to guarantee maximum accuracy) you should use: `((double) hits) / atBats;` since both hits and atBats are integers.
3. League specifications
   - 3.1. There are 30 teams, each with 40 players. (Though only 25 are on a team at a time, this is irrelevant to your task.)
   - 3.2. There are exactly 1200 players in the league.
   - 3.3. Team names are no longer than 3 chars.
4. Stat file format
   - 4.1. Your program will not be dealing with this file directly. It is read and stored by the StatRunner before your class is even created.
   - 4.2. Stat files are Comma Separated Value (csv) files, which means fields in lines are separated by commas.
   - 4.3. Lines 1-30 are the names of the teams in the order they are encoded.
   - 4.4. Lines 31-1230 are the names, and teams of the players. The position of the player in this list is used for all later encodings in the file to save space in the file and StatRunner. Your class will never be aware of this encoding, and will always be dealing with names of players and teams.
   - 4.5. Succeeding lines are a mix of updates and queries.
     - 4.5.1. Queries start with an operation number, then 'Q' followed by a team number, and 10 player numbers in order of batting average, or "tie". A team number of 30, indicates a league query.
     - 4.5.2. Updates start with an operation number, then player number, and then a 1 if there was a hit, else a 0.
5. You may not alter statrunner.cpp, statrunner.h, nor CPUTimer.h.
6. You will find my executable, statrunner.cpp, statrunner.h, CPUTimer.h, a Makefile, a two stat files, CreateStats.out which creates stats files, MLBPlayers.csv which is the data source for CreateStats.out, a barebones stats.cpp, and a barebones stats.h in ~ssdavis/60/p4.
7. Grading

7.1. I will copy statsrunner.cpp, statsrunner.h, and CPUTimer.h into your handin directory, and then call make, so please make sure you handin all necessary files. Students often forget to handin dsexceptions.h, as well as other secondary Weiss files. "handin cs60 p4 *.cpp *.h Makefile" would probably be wise after a successful clean remake.

7.2. The program will be tested using three stats files. The measurements will be the total times from the three runs.

7.2.1. Proper operation: If there are ANY error messages, then the program will receive zero, otherwise it receives a minimum of 15 points.

7.3. Time = 10 points, only possible if proper operation

7.3.1. CPU Time score = min(15, 10 * Sean's CPU / Your CPU)

7.3.2. CPU time may not exceed 100 for one file.

7.3.3. Programs must be compiled without any optimization options. You may not use any precompiled code, including the STL and assembly.

7.4. You may not have any significantly complex or large static, nor significantly complex or large global variables since they would be created before the CPU timer. Note that you may have constants.

8. MakefileStats. The MakefileStats provided contains the minimum needed to create stats.out.

8.1. To ensure that you handin all of the files necessary for your program, you should create a temp directory, and copy only *.cpp, *.h, and MakefileStats into it. Then try to make the program with: `make -f MakefileStats`. Many students have forgotten to handin dsexceptions.h.

9. Suggestions

9.1. Write iteratively. Don't try to write this all at once. Compile and run often during the development process. It is not uncommon for me to re-compile and run after writing a single difficult line of code.

9.2. Keep things simple, and get things running first, and only then use gprof to learn where things are going slowly.

9.3. Use Weiss code where possible. His files are good starting points for any ADT you wish to use.

9.4. Remember to put the authors' names on the first line of stats.h, and to turn in dsexceptions.h if your program needs it!

9.5. You may find qsort() from cstdlib, and memcpy() from cstring handy for implementation.

9.6. If you find you have a bug in the middle of running through a stats file, add an if statement in your code that describes the state of the machine at the time of the bug, and then put a breakpoint at the cout in the if statement. For example, in main()'s operation for-loop you might add the following if the 978th operation was causing trouble:

```
if(i == 977)
   cout << "Help!\n";  // place a breakpoint here.
```

Two runs of stats.out with no errors:

```
[ssdavis@lect1 p4]$ stats.out stats-1.csv
CPU time: 2.08668
[ssdavis@lect1 p4]$ stats.out stats-2.csv
CPU time: 2.09982
[ssdavis@lect1 p4]$
```

```
typedef struct
{
  char name[25];
  char team[4];
} Player;
```

main() of statrunner.cpp:

```
int main(int argc, char** argv) {
  int count;
  char teams[31][4];
  double startTime, endTime;
  Player players[1200], top10[10];
  Operation *operations = new Operation[5000000];
  count = readFile(argv[1], teams, players, operations);
  startTime = getCPUTime();
  Stats *stats = new Stats();

  for(int i = 0; i < 10; i++)
    top10[i].name[0] = top10[i].team[0] = '\0';

  for(int i = 0; i < count; i++)
    if(operations[i].type == UPDATE)
      stats->update((const char*) players[operations[i].parameters[0]].name,
        (const char*) players[operations[i].parameters[0]].team,
        operations[i].parameters[1], i);
    else // QUERY
    {
      stats->query((const char*)teams[operations[i].parameters[0]], top10, i);

      for(int j = 0; j < 10; j++)
        if(operations[i].returnedValues[j] != -1
          && (strcmp(players[operations[i].returnedValues[j]].name,
top10[j].name)
          || strcmp(players[operations[i].returnedValues[j]].team,
top10[j].team)))
          cout << "Bad list for operation "  << i << " position " << j
            << " was " << top10[j].name << ' ' << top10[j].team << " should be "
            << players[operations[i].returnedValues[j]].name << ' '
            << players[operations[i].returnedValues[j]].team << endl;
    } // else QUERY

  endTime = getCPUTime();
  cout << "CPU time: " << endTime - startTime << endl;
  return 0;
}  //main())
```

Retail Sales (25 points, 1 hour)

A large retail company uses typical scanners to read the SKU (Stock Keeping Unit) codes of products at the checkout stand. The sales slip shows the quantity, SKU, product name, and price. You are to write a program that provides information to the check stands, maintains inventory, reports the total sales of ranges of SKUs during the day, and, at the end of the day, provides a list of SKUs that need to be ordered. Initially, the program receives a list of all the product names, SKUs, prices, number in stock, and minimum to have in stock. All information is maintained in RAM.

1. SKUs can contain up 13 digits. There are 593279 different SKUs.
2. Product names contain between 4 and 255 characters.
3. Prices are in pennies, and can be accurately represented in ints. They range from 19 to 9999, and are created randomly.
4. The initial number in stock of a product may be less than the minimum to have in stock. The company will never sell more than is in stock. The number in stock ranges from 0 to 100. The minimum ranges from 1 to 25.
5. Check stand sale provide the SKUs, and number sold. The number sold ranges from 1 to 50, and never exceeds the stock on hand. Your function will return the name and individual price of the product. There is an average of about five sales of each product each day.

6. Ranged reports occur approximately 10 percent of the time.
   6.1. Your class will be provided with two SKUs, in ascending order, and will return the total sales, in pennies, of those items, inclusive.
7. The Below Minimum Report provides a list of SKUs sorted by SKU.
8. Sales files
   8.1. The format of sales filename is: `sales-` <# of transactions> `-` <seed>`.csv.`
   8.2. To minimize their size, all products are referred to by their row in skus2.txt.
   8.3. The first line has <#SKUs>,<# transactions>
   8.4. Second through #SKUs + 1 lines are product initial state: <SKU> <skus2.txt row #>, <onHand>, <minimum>, <price>
   8.5. All but the last following lines are transactions.
      8.5.1. Sales line format: 'S'<SKU>,<row#>,<number>
      8.5.2. Report line format: 'R' <SKU>,<SKU>, <total>
   8.6. Last line is a list of SKU separated by commas for the Below Minimum Report.
9. Grading
   9.1. I will copy salesRunner.cpp, salesRunner.h, and CPUTimer.h into your handin directory, and then call make, so please make sure you handin all necessary files. Students often forget to handin dsexceptions.h, as well as other secondary Weiss files. "handin cs60 p4 *.cpp *.h Makefile" would probably be wise after a successful clean remake.
   9.2. The program will be tested using three sales files, each with 50,000 transactions. The measurements will be the total times from the three runs.
      9.2.1. Proper operation: If there are ANY error messages, then the program will receive zero, otherwise it receives a minimum of 15 points.
   9.3. Time = 10 points, only possible if proper operation
      9.3.1. CPU Time score = min(15, 10 * Sean's CPU / Your CPU)
      9.3.2. CPU time may not exceed 100 for one file.
      9.3.3. Programs must be compiled without any optimization options. You may not use any precompiled code, including the STL and assembly.
   9.4. You may not have any significantly complex or large static, nor significantly complex or large global variables since they would be created before the CPU timer. Note that you may have constants.
10. MakefileSales. The MakefileSales provided contains the minimum needed to create stats.out.
   10.1. To ensure that you handin all of the files necessary for your program, you should create a temp directory, and copy only *.cpp, *.h, and MakefileSales into it. Then try to make the program. Many students have forgotten to handin dsexceptions.h.
11. Suggestions
   11.1. Write iteratively. Don't try to write this all at once. Compile and run often during the development process. It is not uncommon for me to re-compile and run after writing a single difficult line of code.
   11.2. Keep things simple, and get things running first, and only then use gprof to learn where things are going slowly.
   11.3. Use Weiss code where possible. His files are good starting points for any ADT you wish to use.
   11.4. Remember to put the authors' names on the first line of sales.h, and to turn in dsexceptions.h if your program needs it!
   11.5. You may find qsort() from cstdlib, and memcpy() from cstring handy for implementation.
   11.6. If you find you have a bug in the middle of running through a stats file, add an if statement in your code that describes the state of the machine at the time of the bug, and then put a breakpoint at the cout in the if statement. For example, in main()'s operation for-loop you might add the following if the 978th operation was causing trouble:

```
if(i == 977)
    cout << "Help!\n";  // place a breakpoint here.
```

```
[ssdavis@lect1 p4]$ cat sales-10-3.csv
2,10
71698022025,217106,59,17,336
22700043729,52010,79,19,773
R22700043729,22700043729,0
S22700043729,52010,25
S71698022025,217106,15
R22700043729,22700043729,19325
S71698022025,217106,22
R22700043729,71698022025,31757
S71698022025,217106,6
R22700043729,22700043729,19325
R71698022025,71698022025,14448
S71698022025,217106,1
71698022025,
[ssdavis@lect1 p4]$ sales.out sales-10-3.csv
CPU time: 9.174e-06
[ssdavis@lect1 p4]$ sales.out sales-50000-2.csv
CPU time: 1.29256
[ssdavis@lect1 p4]$ salesGold.out sales-50000-2.csv
CPU time: 0.334702
[ssdavis@lect1 p4]$
```

main() of salesRunner.cpp:

```
int main(int argc, char** argv) {
  int numTransactions, *rows, rowCount, belowMinCount, belowMinCount2, price,
    totalSales;
  long long *belowMins, *belowMins2;
  double startTime, endTime;
  const char *ptr;
  Product *products = new Product[594000];
  Transaction *transactions;
  ProductInfo *productInfo;
  readSKUFile(products);
  readSalesFile(&rows, &rowCount, &numTransactions, &transactions, &belowMins,
    &belowMins2, &belowMinCount, argv[1], products, &productInfo);
  startTime = getCPUTime();
  Sales *sales = new Sales(productInfo, rowCount);
  delete [] productInfo;

  for(int i = 0; i < numTransactions; i++)
  {
    if(transactions[i].type == 'S')
    {
      sales->sale(transactions[i].SKU, transactions[i].value, &ptr, &price);

      if(strcmp(ptr, products[transactions[i].productNum1].name))
      cout << "Name error on transaction #" << i << " yours: " << ptr
        << " correct: "  << products[transactions[i].productNum1].name << endl;

      if(price != products[transactions[i].productNum1].price)
        cout << "Price error on transaction #" << i << " yours: "  << price
          << " correct: "  <<  products[transactions[i].productNum1].price <<
endl;
    }  // if sale
    else  // Report type
    {
      sales->report(transactions[i].SKU, transactions[i].SKU2, &totalSales);

      if(totalSales != transactions[i].value)
        cout << "Total sales error on transaction #"  << i << " yours: "
          << totalSales << " correct: "  << transactions[i].value << endl;
    }  // else a report
  } // for each transaction

  sales->belowMinimums(belowMins2, &belowMinCount2);
  endTime = getCPUTime();

  if(belowMinCount != belowMinCount2)
    cout << "Number of minimums error: yours: "  << belowMinCount2
      << " correct: "  << belowMinCount << endl;
  else
    for(int i = 0; i < belowMinCount; i++)
      if(belowMins[i] != belowMins2[i])
        cout << "Below minimum mismatch at #"  << i << " yours: "
          << belowMins2[i] << " correct: "  << belowMins[i] << endl;

  cout << "CPU time: " << endTime - startTime << endl;
  return 0;
} // main())
```