Name: Amuldeep Dhillon

Class: CS116-02 Sha 2017 Spring

Assignment: Lab 5 - Wine and SQL

Date: 5/11/2017

<u>Description:</u> This program will ask the User to enter a Vintage, Price, and Rating range and will use the MySQL database to get the information of wines within the given range along with calculating the average price and number of wines within the range.

<u>Inputs:</u>

- ❖ The Menu selection value
    - ➢ The price range
    - ➢ The rating range
    - ➢ The vintage range

<u>Outputs:</u>

- ❖ The wine info that fall between the given range which includes: the wine name,vintage,rating,price,type
    - ➢ If a score range is selected the results will be ordered from lowest to highest score and then lowest to highest price
    - ➢ If a price range is selected the results will be ordered from lowest to highest price
    - ➢ If a vintage range is selected the results will be ordered from newest to oldest, then score from highest to lowest, and finally price
- ❖ At the end, the number of items in the list is displayed
- ❖ At the end, the average price of the list is displayed

<u>Source Code:</u>

MakeFile:

```
#

#

CC=g++ -std=c++11

#

CFLAGS = -c -Wall -I/usr/include/mysql

LFLAGS = -L/usr/lib/mysql -lmysqlclient




all: lab


lab: printMeFirst.o main.o menu.o scoreComparison.o priceComparison.o vintageComparison.o
\
dbconnect.o wine.o setName.o setVintage.o setRating.o \
setPrice.o setType.o printInfo.o setWineInfo.o
   $(CC) printMeFirst.o main.o menu.o scoreComparison.o priceComparison.o
vintageComparison.o \
   dbconnect.o wine.o setName.o setVintage.o setRating.o \
   setPrice.o setType.o printInfo.o setWineInfo.o -o lab $(LFLAGS)


printMeFirst.o: printMeFirst.cpp lab.h
```

```
   $(CC) $(CFLAGS) printMeFirst.cpp


dbconnect.o: dbconnect.cpp dbconnect.h

   $(CC) $(CFLAGS) dbconnect.cpp


wine.o: wine.cpp lab.h

   $(CC) $(CFLAGS) wine.cpp


setName.o: setName.cpp lab.h

   $(CC) $(CFLAGS) setName.cpp


setVintage.o: setVintage.cpp lab.h

   $(CC) $(CFLAGS) setVintage.cpp


setRating.o: setRating.cpp lab.h

   $(CC) $(CFLAGS) setRating.cpp


setPrice.o: setPrice.cpp lab.h

   $(CC) $(CFLAGS) setPrice.cpp


setType.o: setType.cpp lab.h

   $(CC) $(CFLAGS) setType.cpp
```

```
printInfo.o: printInfo.cpp lab.h
    $(CC) $(CFLAGS) printInfo.cpp

setWineInfo.o: setWineInfo.cpp lab.h
    $(CC) $(CFLAGS) setWineInfo.cpp

menu.o: menu.cpp lab.h
    $(CC) $(CFLAGS) menu.cpp

scoreComparison.o: scoreComparison.cpp lab.h
    $(CC) $(CFLAGS) scoreComparison.cpp

priceComparison.o: priceComparison.cpp lab.h
    $(CC) $(CFLAGS) priceComparison.cpp

vintageComparison.o: vintageComparison.cpp lab.h dbconnect.h
    $(CC) $(CFLAGS) vintageComparison.cpp

main.o: main.cpp lab.h
    $(CC) $(CFLAGS) main.cpp
```

```
clean:

    rm *.o lab


run:

    ./lab "select name, vintage, score, price, type from wineInfo where price > 100"



#end of Makefile
```

Dbconnect.cpp:

```
/*

 *Purpose:

 * Print out programmer's information such as name, class information

 * and date/time the program is run

 *

 * @author Ron Sha

 * @version 1.0 5/1/2017

 *

 * @param name - Amuldeep Dhillon

 * @param courseInfo - Lab 1: CS 116-02 Thursdays

 * @return - none

*/

#include "dbconnect.h"
```

```
MYSQL* mysql_connection_setup(struct connection_details mysql_details)

{

        // first of all create a mysql instance and initialize the variables within

        MYSQL *connection = mysql_init(NULL);


        // connect to the database with the details attached.

        if (!mysql_real_connect(connection,mysql_details.server,

  mysql_details.user, mysql_details.password,

  mysql_details.database, 0, NULL, 0)) {

        printf("Conection error : %s\n", mysql_error(connection));

        exit(1);

        }

        return connection;

}


MYSQL_RES* mysql_perform_query(MYSQL *connection, char *sql_query)

{

  // send the query to the database

  if (mysql_query(connection, sql_query))

  {

        printf("MySQL query error : %s\n", mysql_error(connection));
```

```
            exit(1);
   }

   return mysql_use_result(connection);
}
```

<u>Dbconnect.h:</u>

```
/*
 *Purpose:
 * Contains shared functions and structs for MySQL commands
 *
 * @author Ron Sha
 * @version 1.0 1/27/2017
*/
#ifndef DBCONNECT_H
#define DBCONNECT_H

#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
```

```
#include <iomanip>

using namespace std;

// just going to input the general details and not the port numbers
struct connection_details
{
	char *server;
	char *user;
	char *password;
	char *database;
};

MYSQL* mysql_connection_setup(struct connection_details mysql_details);

MYSQL_RES* mysql_perform_query(MYSQL *connection, char *sql_query);

#endif
```

Lab.h:

```
/*
 *Purpose:
```

```
 * contains Shared classes and fucntions for non-SQL commands

 *

 * NOTE- template functions are in the .h files

 *

 * @author Amuldeep Dhillon

 * @version 1.0 5/6/2017

*/


#pragma once


#include <iostream>

#include <typeinfo>

#include <iomanip>

#include <limits>

#include <sstream>

using namespace std;


void printMeFirst(string, string);

void menu(int &);

void priceComparison(string &,string &,string &);

void scoreComparison(string &,string &,string &);

void vintageComparison(string &,string &, string &);
```

```
template< typename T > class List; // forward declaration

class Wine;


template< typename NODETYPE  >

class ListNode

{


public:

   friend class List< NODETYPE >; // make List a friend

   ListNode( const NODETYPE & ); // constructor

   NODETYPE getData() const; // return the data in the node


/*

 *Purpose:sets the next pointer to the next pointer

 *

 *

 * @author Amuldeep Dhillon

 * @version 1.0 5/6/2017

 *

 * @param nPtr- ListNode pointer

 * @return - none
```

```
*/
  void setNextPtr( ListNode *nPtr )
  {
        nextPtr = nPtr;
  }

/*
 *Purpose: retrieve the next pointer in the linked list
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param - none
 * @return - the next pointer
*/
  ListNode *getNextPtr() const
  {
        return nextPtr;
  }

private:
  NODETYPE data;

  int key;

  ListNode *nextPtr;

};




/*

 *Purpose:Constructs a Linked list Node

 *

 *

 * @author Ron Sha

 * @version 1.0 5/1/2017

 *

 * @param info- the data we wish to store in the linked list node

 * @return - none

*/

template< typename NODETYPE >

ListNode< NODETYPE >::ListNode( const NODETYPE &info )

{

  data = info;

  nextPtr = 0;

}
```

```
/*
 *Purpose:returns a copy of the data in the node
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param -none
 * @return - the data inside a node in the linked list
*/

template< typename NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const
{
   return data;
}



template< typename NODETYPE  >
class List
{
```

```
public:
   List();
   List( const List< NODETYPE > & );
   ~List();

   void insertAtFront( const NODETYPE &, int );
   void insertAtBack( const NODETYPE &, int );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
   void printPtrFunc(   );
   NODETYPE * getInfo(int myKey);

/*
 *Purpose:get the first pointer in a linked list
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param - non
 * @return - the first pointer
*/

   ListNode< NODETYPE >  *getFirstPtr() const
   {
           return firstPtr;
   }

 protected:
   ListNode< NODETYPE > *firstPtr;
   ListNode< NODETYPE > *lastPtr;

   ListNode< NODETYPE > *getNewNode( const NODETYPE &, int );
};

/*
 *Purpose:Create an empty Linked List
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param nPtr- ListNode pointer

 * @return - none

*/

template< typename NODETYPE >

List< NODETYPE >::List()

{

   firstPtr = lastPtr = 0;

}


/*

 *Purpose:copies a currently existing linked list

 *

 *

 * @author Ron Sha

 * @version 1.0 5/1/2017

 *

 * @param copy - The list we want to copy

 * @return - none

*/

template< typename NODETYPE >

List< NODETYPE >::List( const List<NODETYPE> &copy )

{
   firstPtr = lastPtr = 0; // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   // insert into the list
   while ( currentPtr != 0 )
   {
           insertAtBack( currentPtr->data );
           currentPtr = currentPtr->nextPtr;
   } // end while
}

/*
 *Purpose:Destroys the List created by the constructors
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param - none
 * @return - none
*/
```

```
template< typename NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() )
   {

            ListNode< NODETYPE > *currentPtr = firstPtr;
            ListNode< NODETYPE > *tempPtr;

            while ( currentPtr != 0 )
            {
            tempPtr = currentPtr;
            currentPtr = currentPtr->nextPtr;
            delete tempPtr;
            }
   }

}

/*
 *Purpose:Inserts a node at the front of the list
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param value - the data we want to add to the linked list
 * @param key - unique key to distinguish data
 * @return - none
*/
template< typename NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value,
  int key)
{
   ListNode<NODETYPE> *newPtr = getNewNode( value, key );


   if ( isEmpty() )
         firstPtr = lastPtr = newPtr;
   else
   {
         newPtr->nextPtr = firstPtr;
         firstPtr = newPtr;
   }
}
```

```
/*
 *Purpose:Insert a node at the back of the list
 *
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param value - the data we want to add to the list
 * @param key - the unique key used to distinguish data
 * @return - none
*/
// Insert a node at the back of the list
template< typename NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value,
  int key)
{

  ListNode<NODETYPE> *newPtr = getNewNode( value, key );

  if(isEmpty())
    firstPtr = lastPtr = newPtr;
  else
  {
      lastPtr->nextPtr = newPtr;
      lastPtr = newPtr;
  }







}


/*
 *Purpose:removes a node from the front of a linked list
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param value - the location we want the removed data to be saved
 * @return - true
*/
```

```
template< typename NODETYPE >

bool List< NODETYPE >::removeFromFront( NODETYPE &value )

{

  if ( isEmpty() ) // List is empty

          return false; // delete unsuccessful

  else

  {

          ListNode< NODETYPE > *tempPtr = firstPtr;



          if ( firstPtr == lastPtr )

          firstPtr = lastPtr = 0;

          else

          firstPtr = firstPtr->nextPtr;



          value = tempPtr->data;



          delete tempPtr;

          return true;

  }

}


/*
```

```
 *Purpose:remove a node from the back of a linked list
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param value - the location the node data will be placed once deleted
 * @return - true
*/
template< typename NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{



   if ( isEmpty() )
         return false; // delete unsuccessful
   else
   {
         ListNode< NODETYPE > *tempPtr = lastPtr;

         if ( firstPtr == lastPtr )
```

```
            firstPtr = lastPtr = 0;

            else

            {


            ListNode< NODETYPE > *currentPtr = firstPtr;


while(currentPtr->nextPtr != lastPtr)

            {

                        currentPtr = currentPtr->nextPtr;

             }

             lastPtr=currentPtr;

             currentPtr->nextPtr=NULL;


            }


            value = tempPtr->data;

            delete tempPtr;

            return true;

   }

}


/*
```

```
 *Purpose:Check if the list is Empty

 *

 *

 * @author Ron Sha

 * @version 1.0 5/1/2017

 *

 * @param - none

 * @return - true if list is empty, false if list is false

*/

template< typename NODETYPE >

bool List< NODETYPE >::isEmpty() const

{

   return firstPtr == 0;

}


/*

 *Purpose:return a pointer to a newly allocated node

 *

 *

 * @author Ron Sha

 * @version 1.0 5/1/2017

 *
 * @param value - the data we want to add to the new node
 * @return - the pointer to the new node
*/
template< typename NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode(
   const NODETYPE &value, int)
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   return ptr;
}


/*
 *Purpose:Display the contents of the List
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param - none
 * @return - none
*/
template< typename NODETYPE >
```

```
void List< NODETYPE >::print() const

{

  if ( isEmpty() )

  {

          cout << "The list is empty\n\n";

          return;

  }


  ListNode< NODETYPE > *currentPtr = firstPtr;




  while ( currentPtr != 0 )

  {

          int i;

          string s;

          double d;

          char c;

          if (typeid(currentPtr->data).name() == typeid(i).name() ||

          typeid(currentPtr->data).name() == typeid(d).name() ||

          typeid(currentPtr->data).name() == typeid(s).name() ||

          typeid(currentPtr->data).name() == typeid(c).name())

          {
            cout << currentPtr->data << ' ';

            }

            else {

            cout <<"Can't print - Not a simple data type (int, string, char, double)\n";

            }

            currentPtr = currentPtr->nextPtr;

    }


    cout << "\n\n";

}


/*

 *Purpose:Retrieve the contents of the list

 *

 *

 * @author Ron Sha

 * @version 1.0 5/1/2017

 *

 * @param key - the unique key for the data

 * @return - NULL if empty or Data in the node

*/

// Display the contents of the List
```

```
template< typename NODETYPE >

NODETYPE * List< NODETYPE >::getInfo(int myKey)

{

   if ( isEmpty() ) // empty list

   {

           //cout << "The list is empty\n\n";

           return NULL;

   } // end if



   ListNode< NODETYPE > *currentPtr = firstPtr;



   //cout << "The list is: ";



   while ( currentPtr != 0 ) // display elements in list

   {

           if (currentPtr->key == myKey )  // found

           return (& currentPtr->data);



           currentPtr = currentPtr->nextPtr;

   } // end while



   return NULL;  // can't find
```

```
}



/*
 *Purpose:Prints the list out
 *
 *
 * @author Ron Sha
 * @version 1.0 5/1/2017
 *
 * @param nodeList - The list we want to print out
 * @return - none
*/
template< typename NODETYPE >
void printNoteInfo (  List< NODETYPE > & nodeList)
{
  NODETYPE *wp;
  wp = (NODETYPE *) nodeList.getInfo(0); //get node based on key
  //wp->printInfo();



  ListNode< NODETYPE > *currentPtr;
```

```
   currentPtr =  nodeList.getFirstPtr();


   //cout << "The node list is: \n";

   //print out all the info in linked list

   while ( currentPtr != 0 ) // display elements in list

   {

          wp = (NODETYPE *) currentPtr; //convert to correct data type

          wp->printInfo();

          currentPtr = currentPtr->getNextPtr();

   } // end while

}



/*

 *

 * Wine funtions are initialized in their own files

 *

 *

*/

class Wine
```

```
{
public:

   Wine() {  }

   Wine(string wine_name,string wine_type, int wine_year,

            int wine_rating, double wine_price);

   void setInfo(string wine_name,string wine_type, int wine_year,

            int wine_rating, double wine_price);

   ~Wine() { }

   void setName(string wine_name);

   void setVintage(int wine_year);

   void setRating(int wine_rating);

   void setPrice(double wine_price);
```

```
    void setType(string wine_type);

            void printInfo();

    private:

            string wineName, wineType;

            int wineYear, wineRating;

            double winePrice;

};
```

Wine.cpp:

```
/*
 * Purpose:sets the wine's name, type, year, rating, and price manually
 * respective to each parameter at the constructor level
 *
 * @author: Amuldeep Dhillon

 * @version: 1.0 3/7/2016

 *

 * @param wine's name: The name you want the wine to have

 * @param wine's type: The type you want the wine to have

 * @param wine's vintage: The vintage you want the wine to have

 * @param wine's rating: The rating you want the wine to have

 * @param wine's price: The price you want the wine to have

 *

 * @return:none

*/

#include "lab.h"


Wine::Wine(string wine_name,string wine_type, int wine_year,

        int wine_rating, double wine_price){

                wineName = wine_name;

                wineType = wine_type;

                wineYear = wine_year;

                wineRating = wine_rating;

                winePrice = wine_price;

}
```

setWineInfo.cpp:

```
#include "lab.h"


/*
 * Purpose:sets the wine's name, type, year, rating, and price manually
 * respective to each parameter
 *
 * @author: Amuldeep Dhillon
 * @version: 1.0 3/7/2016
 *
 * @param wine's name: The name you want the wine to have
 * @param wine's type: The type you want the wine to have
 * @param wine's vintage: The vintage you want the wine to have
 * @param wine's rating: The rating you want the wine to have
 * @param wine's price: The price you want the wine to have
 *
 * @return:none
*/



void Wine::setInfo(string wine_name,string wine_type, int wine_year,
		int wine_rating, double wine_price){
```

```
                    wineName = wine_name;

                    wineType = wine_type;

                    wineYear = wine_year;

                    wineRating = wine_rating;

                    winePrice = wine_price;

}
```

setName.cpp:

```
/*

 *Purpose:set the Name of the Wine

 *

 * @author Amuldeep Dhillon

 * @version 1.0 5/6/2017

 *

 * @param wine_name = the name we want the wine to have

 * @return - none

*/

#include "lab.h"


void Wine::setName(string wine_name){

   wineName = wine_name;

}
```

setPrice.cpp:

```
/*
 *Purpose:set the Price of the Wine
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param wine_price = the price we want the wine to have
 * @return - none
*/
#include "lab.h"


void Wine::setPrice(double wine_price){
    winePrice = wine_price;
}
```

setRating.cpp:

```
/*
 *Purpose:set the Rating of the Wine
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param wine_rating = the rating we want the wine to have
 * @return - none
*/
#include "lab.h"


void Wine::setRating(int wine_rating){
    wineRating = wine_rating;
}
```

setType.cpp

```
/*
 *Purpose:set the Type of the Wine
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param wine_type = the type we want the wine to have
 * @return - none
*/
#include "lab.h"
```

```
void Wine::setType(string wine_type){
   wineType = wine_type;
}
```

setVintage.cpp:

```
/*
 *Purpose:set the Vintage of the Wine
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param wine_vintage = the vintage we want the wine to have
 * @return - none
*/
#include "lab.h"

void Wine::setVintage(int wine_year){
   wineYear = wine_year;
}
```

printInfo.cpp:

```
/*
 *Purpose:Print the data in the Wine class objects
 *
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param - none
 * @return - none
*/
#include "lab.h"



void Wine::printInfo(){


   cout << left << setw(30) << setfill(' ') << wineName << setw(15) << setfill(' ')
          << wineYear << setw(15) << setfill(' ') << wineRating << setw(15) << setfill(' ')
          << winePrice << setw(15) << setfill(' ') << wineType <<endl;
}
```

printMeFirst.cpp:

```
/*
```

```
 *Purpose:

 * Print out programmer's information such as name, class information

 * and date/time the program is run

 *

 * @author Ron Sha

 * @version 1.0 1/27/2017

 *

 * @param name - Amuldeep Dhillon

 * @param courseInfo - Lab 1: CS 116-02 Thursdays

 * @return - none

*/



#include "lab.h"

void printMeFirst(std::string name, std::string courseInfo)


{


std::cout <<" Program written by: "<< name << std::endl; // put your name here


std::cout <<" Course info: "<< courseInfo << std::endl;
time_t now = time(0);  // current date/time based on current system

char* dt = ctime(&now); // convert now to string for

  std::cout << " Date: " << dt << std::endl;


}
```

Menu.cpp:

```
/*
 *Purpose:Display a menu for the Wine Database
 *
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param input - The input that the user inputs
 * @return - none
*/
#include "lab.h"


void menu(int &input){
    cout << "Please select how you would like to sort your wines\n";
    cout << "1 - By Rating\n";
    cout << "2 - By Price\n";
    cout << "3 - By Vintage\n";
    cout << "4 - Exit\n";
    cout << "Enter your Choice: ";
    while(!(cin >> input) || input > 4 || input < 1){
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Error: Please enter a valid Input: ";
    }

}
```

<u>priceComparison.cpp:</u>

```
/*
 *Purpose:Takes a lower and upper bound for prices and creates the MySQL command
 *           to retrieve data, the number of data pieces, and average price
 *
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param command - the string for the SQL command to find the data
 * @param count - the string for the SQL command to count the data
 * @param average - the string for the SQL command to average the prices
 * @return - none
*/
#include "lab.h"


void priceComparison(string & command,string & count,string & average){
   int bottom, top, temp;
   cout << "Please enter a Bottom Price Value: " << endl;
   while(!(cin >> bottom)){
          cin.clear();
          cin.ignore(numeric_limits<streamsize>::max(), '\n');
          cout << "Error: Please enter a valid Input: ";
   }
   cout << "Please enter a Top Price Value: " << endl;
   while(!(cin>>top)){
          cin.clear();
          cin.ignore(numeric_limits<streamsize>::max(), '\n');
          cout << "Error: Please enter a valid Input: ";
   }
```

```
if(bottom > top){

        cout << "Bottom is greater than Top, switching values\n";

        temp = top;

        top = bottom;

        bottom = temp;

}


string part1 = "select name, vintage, score, price, type from wineInfo where price >= ";

string part2 = " and price <= ";

string part3 = " order by price";

stringstream convert;

convert << part1 << bottom << part2 << top << part3;

command = convert.str();


string part4 = "select count(name) from wineInfo where price >= ";

string part5 = " and price <= ";

stringstream counter;

counter << part4 << bottom << part5 << top;

count = counter.str();


string part6 = "select round(avg(price),2) from wineInfo where price >= ";

string part7 = " and price <= ";
```

```
    stringstream averager;

    averager << part6 << bottom << part7 << top;

    average = averager.str();

}
```

scoreComparison.cpp:

```
/*

 *Purpose:Takes a lower and upper bound for scores and creates the MySQL command

 *           to retrieve data, the number of data pieces, and average price

 *

 *

 * @author Amuldeep Dhillon

 * @version 1.0 5/6/2017

 *

 * @param command - the string for the SQL command to find the data

 * @param count - the string for the SQL command to count the data

 * @param average - the string for the SQL command to average the prices

 * @return - none

*/

#include "lab.h"


void scoreComparison(string & command,string & count,string & average){
```

```
int bottom, top, temp;

cout << "Please enter a Bottom Rating Value: " << endl;

while(!(cin >> bottom) || bottom < 0 || bottom > 100){

        cin.clear();

        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        cout << "Error: Please enter a valid Input: ";

}

cout << "Please enter a Top Rating Value: " << endl;

while(!(cin>>top) || top < 0 || top > 100){

        cin.clear();

        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        cout << "Error: Please enter a valid Input: ";

}

if(bottom > top){

        cout << "Bottom is greater than Top, switching values\n";

        temp = top;

        top = bottom;

        bottom = temp;

}


string part1 = "select name, vintage, score, price, type from wineInfo where score >= ";

string part2 = " and score <= ";
```

```
    string part3 = " order by score,price";

    stringstream convert;

    convert << part1 << bottom << part2 << top << part3;

    command = convert.str();


    string part4 = "select count(name) from wineInfo where score >= ";

    string part5 = " and score <= ";

    stringstream counter;

    counter << part4 << bottom << part5 << top;

    count = counter.str();


    string part6 = "select round(avg(price),2) from wineInfo where score >= ";

    string part7 = " and score <= ";

    stringstream averager;

    averager << part6 << bottom << part7 << top;

    average = averager.str();

}
```

vintageComparison.cpp:

```
/*

 *Purpose:Takes a lower and upper bound for vintage and creates the MySQL command

 *          to retrieve data, the number of data pieces, and average price
 *
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param command - the string for the SQL command to find the data
 * @param count - the string for the SQL command to count the data
 * @param average - the string for the SQL command to average the prices
 * @return - none
*/
#include "lab.h"


void vintageComparison(string & command, string & count, string & average){
   int bottom, top, temp;
   cout << "Please enter an Oldest Vintage: " << endl;
   while(!(cin >> bottom)){
          cin.clear();
          cin.ignore(numeric_limits<streamsize>::max(), '\n');
          cout << "Error: Please enter a valid Input: ";
   }
   cout << "Please enter a Newest Vintage: " << endl;
   while(!(cin>>top)){
        cin.clear();

        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        cout << "Error: Please enter a valid Input: ";

}

if(bottom > top){

        cout << "Bottom is greater than Top, switching values\n";

        temp = top;

        top = bottom;

        bottom = temp;

}


string part1 = "select name, vintage, score, price, type from wineInfo where vintage >= ";

string part2 = " and vintage <= ";

string part3 = " order by vintage DESC,score DESC, price ASC";

stringstream convert;

convert << part1 << bottom << part2 << top << part3;

command = convert.str();


string part4 = "select count(name) from wineInfo where vintage >= ";

string part5 = " and vintage <= ";

stringstream counter;

counter << part4 << bottom << part5 << top;
```

```
    count = counter.str();


    string part6 = "select round(avg(price),2) from wineInfo where vintage >= ";
    string part7 = " and vintage <= ";
    stringstream averager;
    averager << part6 << bottom << part7 << top;
    average = averager.str();
}
```

Main.cpp:

```
/*
 *Purpose:starts and calls all other functions
 *
 *
 * @author Amuldeep Dhillon
 * @version 1.0 5/6/2017
 *
 * @param argc - unused
 * @param argv[] - unused
 * @return - none
*/
```

```
#include "lab.h"

#include "dbconnect.h"




using namespace std;









/*

 argv[1] - put sql command in argv[1], otherwise, just

 use sql "show tables"

*/

int main(int argc, char* argv[])

{

    printMeFirst("Amuldeep Dhillon", "Lab 5 - CS116");

    int choice = 0;

    string sqlCommand,countCommand, avgCommand;

  MYSQL *conn;         // the connection

  MYSQL_RES *res;    // the results

  MYSQL_ROW row;    // the results row (line by line)
  struct connection_details mysqlD;

  mysqlD.server = (char *)"localhost";  // where the mysql database is

  mysqlD.user = (char *)"root";           // the root user of mysql

  mysqlD.password = (char *)"password"; // the password of the root user in mysql

  mysqlD.database = (char *)"mysql";    // the databse to pick


  // connect to the mysql database

  conn = mysql_connection_setup(mysqlD);

 cout << "Welcome to the Wine Database \n";

 while(choice != 4){

    menu(choice);

    if(choice == 1){

          scoreComparison(sqlCommand,countCommand,avgCommand);

    }

    if(choice == 2){

          priceComparison(sqlCommand,countCommand,avgCommand);

    }

    if(choice == 3){

          vintageComparison(sqlCommand,countCommand,avgCommand);

    }

    if(choice == 4){

          cout << "Good Bye\n";
```

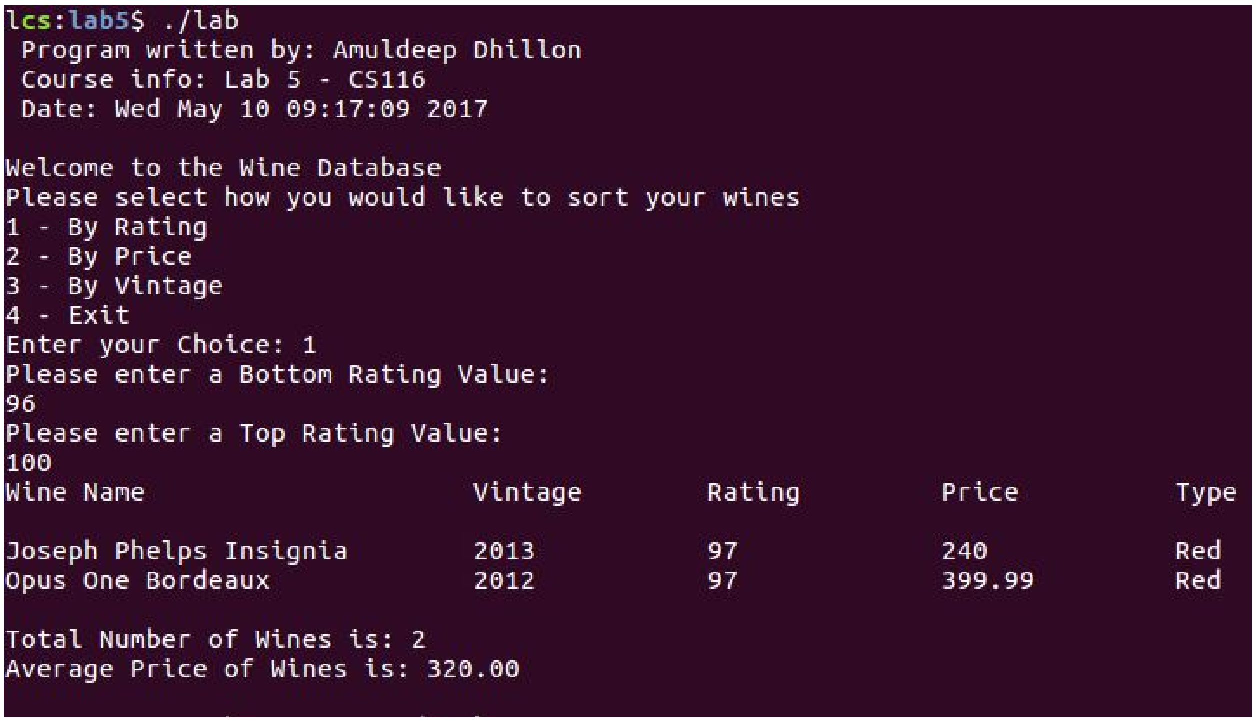
```
            break;
   }


// assign the results return to the MYSQL_RES pointer
/*
if (argc < 2)
{
        cout << "argv[0]: " << argv[0] << endl;
        printf("\nUsage: %s  \"SQL statement here\"\n", argv[0]);
        printf("if no argument given, default is:\n %s show tables\n", argv[0]);
        res = mysql_perform_query(conn, (char *)"show tables");
        printf("MySQL Tables in mysql database:\n");
}
* */
//else
//{
        // use wine database
        res = mysql_perform_query(conn, (char *)"use wine");
        //cout << "argv[0]: " << argv[0] << endl << endl;
        //cout << "argv[1]: " << argv[1] << endl << endl;
        res = mysql_perform_query(conn, ((char*)(sqlCommand.c_str())));
        /*
        * you need to print out the header.  Make sure it it
        * nicely formated line up.  Modify the cout statement
        * below so the header is nicely line up.  Hint: use left and setw
        *
        * WineName   Vitange  Rating  Price  Type
        * */
        cout << left << setw(30) << setfill(' ') << "Wine Name"
                << setw(15) << setfill(' ') << "Vintage"
                                                << setw(15) << setfill(' ') << "Rating"
                                                << setw(15) << setfill(' ') << "Price"
                                                << setw(15) << setfill(' ') << "Type"
  << endl << endl;
//}
  int i=0;
  Wine w;
  List < Wine > wineList;
while ((row = mysql_fetch_row(res)) !=NULL)
{
        //if (argc < 2) {
        //   printf("%s\n", row[0]);  // only print out 1st column
        //}
        //else
```

```
    //{



        w.setName(row[0]);

        w.setVintage(stoi(row[1]));

        w.setRating(stoi(row[2]));

        w.setPrice(stod(row[3]));

        w.setType(row[4]);


        wineList.insertAtBack(w,i);




        /*

        // print out each row of the data extracted from

            // MySQL database

            // Make sure the output is line up with the header

            // Hint: use left and setw

            //


cout << left << setw(30) << setfill(' ') << row[0]   // coulumn (field) #1 - Wine Name

 << setw(15) << setfill(' ') << row[1] // field #2 - Vintage
```

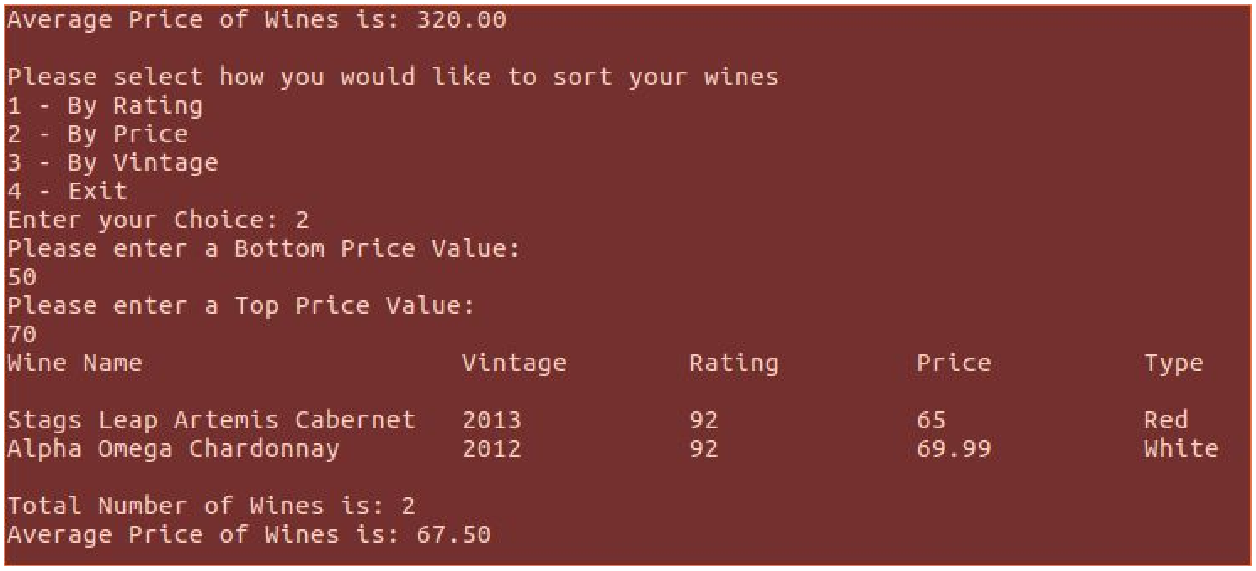
```
    << setw(15) << setfill(' ') << row[2] // field #3 - Rating

   << setw(15) << setfill(' ') << row[3] // field #4 - Price

    << setw(15) << setfill(' ') << row[4] // field #5 - Wine type

    << endl; // field #7 - UPC

    */

        //}




}

printNoteInfo(wineList);

cout << "\nTotal Number of Wines is: ";

res = mysql_perform_query(conn, ((char*)(countCommand.c_str())));

row = mysql_fetch_row(res);

cout << row[0] << endl;



mysql_free_result(res);

cout << "Average Price of Wines is: ";

res = mysql_perform_query(conn, ((char*)(avgCommand.c_str())));

row = mysql_fetch_row(res);

cout << row[0] << endl << endl;



mysql_free_result(res);
```

```
}



  /* clean up the database link */

  mysql_close(conn);



  return 0;

}
```

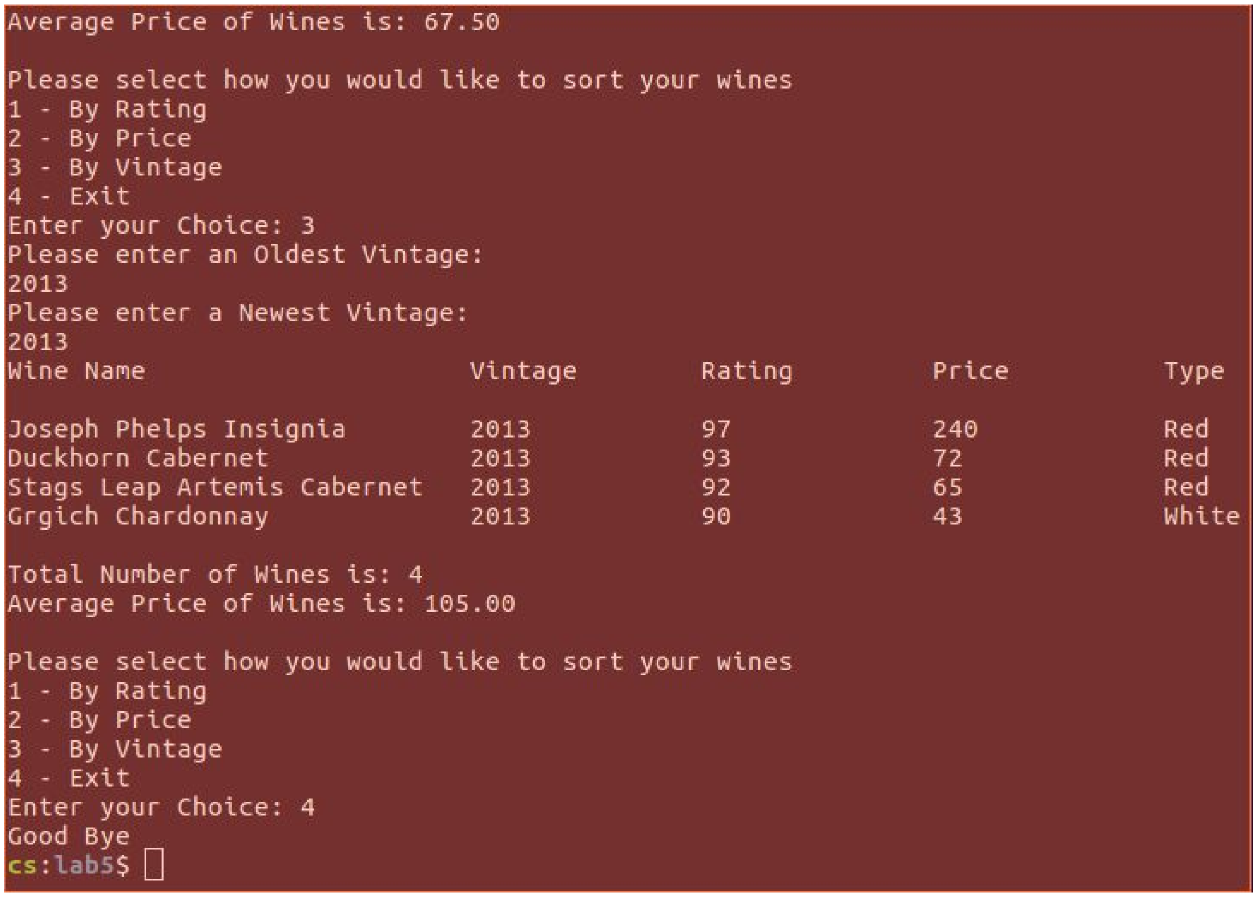
Test Screenshots:



```
lcs:lab5$ ./lab
 Program written by: Amuldeep Dhillon
 Course info: Lab 5 - CS116
 Date: Wed May 10 09:17:09 2017

Welcome to the Wine Database
Please select how you would like to sort your wines
1 - By Rating
2 - By Price
3 - By Vintage
4 - Exit
Enter your Choice: 1
Please enter a Bottom Rating Value:
96
Please enter a Top Rating Value:
100
Wine Name                    Vintage        Rating        Price         Type

Joseph Phelps Insignia       2013           97            240           Red
Opus One Bordeaux            2012           97            399.99        Red

Total Number of Wines is: 2
Average Price of Wines is: 320.00
```



```
Average Price of Wines is: 320.00

Please select how you would like to sort your wines
1 - By Rating
2 - By Price
3 - By Vintage
4 - Exit
Enter your Choice: 2
Please enter a Bottom Price Value:
50
Please enter a Top Price Value:
70
Wine Name                      Vintage        Rating         Price          Type

Stags Leap Artemis Cabernet    2013           92             65             Red
Alpha Omega Chardonnay         2012           92             69.99          White

Total Number of Wines is: 2
Average Price of Wines is: 67.50
```



```
Average Price of Wines is: 67.50

Please select how you would like to sort your wines
1 - By Rating
2 - By Price
3 - By Vintage
4 - Exit
Enter your Choice: 3
Please enter an Oldest Vintage:
2013
Please enter a Newest Vintage:
2013
Wine Name                      Vintage        Rating         Price          Type

Joseph Phelps Insignia         2013           97             240            Red
Duckhorn Cabernet              2013           93             72             Red
Stags Leap Artemis Cabernet    2013           92             65             Red
Grgich Chardonnay              2013           90             43             White

Total Number of Wines is: 4
Average Price of Wines is: 105.00

Please select how you would like to sort your wines
1 - By Rating
2 - By Price
3 - By Vintage
4 - Exit
Enter your Choice: 4
Good Bye
cs:lab5$
```