CS116 – Lab 4:

cs116 lab4 is to practice and get familiar with using classes, multiple files, using pointers, mySQL, LinkedList, and other methods to access members between different classes. It is to help us understand how to take information from a database in mySQL, pull the data into a LinkedList, read those results from our LinkedList, and then manipulating the mySQL statements to print the desired values from our mySQL database.

In mySQL, we use a database called wine and select columns and rows from wineInfo to get the information. While iterating through all the rows given by the mySQL statement, we convert those rows to match the parameters within our WineClass so we can set the information for our WineList(Name, Vintage, Score, Price, Type). After doing so, we use pointers in our List class to insert information to the back of our list and then reading off the desired output from LinkedList.

For the remaining portion of the lab, we use mySQL statements to display the user's desired output. Implementing everything in a menu selection, we ask the user if they'd like to sort by score and price; only price; and vintage and score. Score and price reads off LinkedList but also requires user input of desired score range and assigns that range in the mySQL statement, where after the results are placed in LinkedList and printed out. The other two function the same by asking for user's desired range but only prints the results read from mySQL.

Source code(wine.cpp, wine.h, printMeFirst.cpp, printMeFirst.h, LinkedList.h, LinkedList.hpp, dbconnect.cpp, dbconnect.h, main.cpp)

```
// wine.h

#include <iostream>

#include <typeinfo>

#include <iomanip>



#ifndef WINE_H

#define WINE_H



using namespace std;



class Wine

{

	public:

		Wine();

		Wine(string n, int v, int s, double p, string t);

		void setInfo(string n, int v, int s, double p, string t);

		void setPrice(double p);

		string getName() const;

		int getPrice() const;

		void printInfo();

		//void printWineinfo(List< Wine > & wineList);



	private:

		string name;

		int vintage;

		int score;
```

```
                double price;
                string type;
};

#endif
// wine.cpp
#include "wine.h"
/**

  Purpose:

    Constructor.

  @author Ron Sha

  @version 1.0 1/27/2017



  @param - none

  @return - none

*/
Wine::Wine()
{
}
/**

  Purpose:
```

```
    Assign the information on Wine class

  @author Ron Sha

  @version 1.0 1/27/2017



  @param n - wineName

  @param v - wineVintage

  @param s - wineScore

  @param p - winePrice

  @param t - wineType

  @return - none

*/
Wine::Wine(string n, int v, int s, double p, string t)
{
        name = n;
        vintage = v;
        score = s;
        price = p;
        type = t;
}
/**
```

```
   Purpose:

      Assign the information on Wine class

   @author Ron Sha

   @version 1.0 1/27/2017




   @param n - wineName

   @param v - wineVintage

   @param s - wineScore

   @param p - winePrice

   @param t - wineType

   @return - none

*/
void Wine::setInfo(string n, int v, int s, double p, string t)
{
        name = n;
        vintage = v;
        score = s;
        price = p;
        type = t;
}
```

```
/**

   Purpose:

      Set Wine Price

   @author Ron Sha

   @version 1.0 1/27/2017



   @param p - winePrice

   @return - none

*/
void Wine::setPrice(double p)
{
        price = p;
}
/**

   Purpose:

      Retrieve and return wine Name

   @author Ron Sha

   @version 1.0 1/27/2017
```

```
   @param n - wineName

   @return - name

*/
string Wine::getName() const
{
        return name;
}
/**

   Purpose:

      Return and retrieve wine Price

   @author Ron Sha

   @version 1.0 1/27/2017



   @param p - winePrice

   @return - price

*/
int Wine::getPrice() const
{
        return price;
```

```
}
/**

   Purpose:

      Print out the Wines in our class list.

   @author Kenneth Surban

   @version 1.0 5/11/2017



   @param - none

   @return - none

*/
void Wine::printInfo()
{
            cout << setw(32) << left << name << setfill(' ') // coulumn (field) #1 - Wine Name
     << setw(15) << vintage << setfill(' ') // field #2 - Vintage
     << setw(15) << score << setfill(' ') // field #3 - Rating
    << setw(13) << price << setfill(' ') // field #4 - Price
    << setw(10) << type << setfill(' ') // field #5 - Wine type
    << endl;
}

/*
void printWineinfo(List< Wine > & wineList)
{
```

```
	Wine * f;
	f = (Wine *) wineList.getInfo(0);
	f->printInfo();

	ListNode< Wine > *currentPtr;

	currentPtr = personList.getFirstPtr();

	cout << "The Wine list is: \n";

	while( currentPtr != 0 )
	{
		f = (Wine *) currentPtr;
		f->printInfo();
		currentPtr = currentPtr->getNextptr();
	}
}
*/
// printMeFirst.h
#include <iostream>
#include <string>
#include <iomanip>
#include <ctime>
#ifndef PRINTMEFIRST_H
#define PRINTMEFIRST_H

using namespace std;

void printMeFirst(string name, string courseInfo);

#endif
```

```
// printMeFirst.cpp
#include "printMeFirst.h"

/**

   Purpose:

      Print out the 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 -  the name of the programmer

   @param courseInfo -  the name of the course

   @return - none

*/

void printMeFirst(string name, string courseInfo)

{

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

```
cout <<" Course info: "<< courseInfo << endl;

time_t now = time(0);  // current date/time based on current system

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

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

}
// LinkedList.h
#include <iostream>
#include <typeinfo>

#ifndef LINKEDLIST_H
#define LINKEDLIST_H

using namespace std;

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

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

   // set nextPtr to nPtr
   void setNextPtr( ListNode *nPtr )
```

```
   {
      nextPtr = nPtr;
   } // end function setNextPtr

   // return nextPtr
   ListNode *getNextPtr() const
   {
      return nextPtr;
   } // end function getNextPtr

private:
   NODETYPE data; // data
   int key; // used for key for the list
   ListNode *nextPtr; // next node in the list
}; // end class ListNode

template< typename NODETYPE  >
class List
{
public:
   List(); // default constructor
   List( const List< NODETYPE > & ); // copy constructor
   ~List(); // destructor

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

```
   void printNoteInfo(   );
   NODETYPE * getInfo(int myKey);
     // return nextPtr
   ListNode< NODETYPE >  *getFirstPtr() const
   {
     return firstPtr;
   } // end function getNextPtr

 protected:
   ListNode< NODETYPE > *firstPtr; // pointer to first node
   ListNode< NODETYPE > *lastPtr; // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE &, int );
}; // end class template List

#endif
//LinkedList.hpp

#ifndef LINKEDLIST_HPP
#define LINKEDLIST_HPP

#include "LinkedList.h"



/**

   Purpose:

      Assign node data the address of node info which is assigned
      to the address of the variables where we set our info for our class
```

```
      and set nextPtr to NULL.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param &info - node address, assigned the content of our class setInfo
                  function.

   @return - none

*/
// constructor
template< typename NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
} // end constructor
/**

   Purpose:

      Get data and return it

   @author Ron Sha

   @version 1.0 1/27/2017
```

```
   @param - none

   @return - data

*/
// return a copy of the data in the node
template< typename NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const
{
   return data;
} // end function getData
/**

   Purpose:

      Constructor for list. To assign firstPtr and lastPtr to NULL.

   @author Ron Sha

   @version 1.0 1/27/2017




   @param - none

   @return - none

*/
// default constructor
```

```
template< typename NODETYPE >
List< NODETYPE >::List()
{
   firstPtr = lastPtr = 0;
} // end constructor
/**

   Purpose:

      insertAtBack our data by using currentPtr->data and then by pointing
      it to the nextPtr via currentPtr->nextPtr.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param - none

   @return - none

*/
// copy constructor
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
} // end List copy constructor
/**

   Purpose:

      Desconstructor. Print "Destroying nodes..."" when program exits
      and destroy nodes by using tempPtr. Since is assigned the value pointed by
      currentPtr, and when currentPtr points to data. We can delete it by saying
      delete tempPtr.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param - none

   @return - none

*/
// destructor
template< typename NODETYPE >
List< NODETYPE >::~List()
{
```

```
   if ( !isEmpty() ) // List is not empty
   {
      cout << "Destroying nodes ...\n";

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

      while ( currentPtr != 0 ) // delete remaining nodes
      {
         tempPtr = currentPtr;
//         cout << tempPtr->data << ' ';
         currentPtr = currentPtr->nextPtr;
         delete tempPtr;
      } // end while
   } // end if

   cout << "\nAll nodes destroyed\n\n";
} // end destructor/**
/*
   Purpose:

       Insert our desired node at the front of the list.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param &value - The node we are adding to the the front of the list
```

```
   @param key - ID key for list

   @return - none

*/
// Insert a node at the front of the list
template< typename NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value, int key)
{
   ListNode<NODETYPE> *newPtr = getNewNode( value, key );

   if ( isEmpty() ) // List is empty
      firstPtr = lastPtr = newPtr;
   else // List is not empty
   {
      newPtr->nextPtr = firstPtr;
      firstPtr = newPtr;
   } // end else
} // end function insertAtFront
/**

   Purpose:

      Insert a node at the back of the list.

   @author Ron Sha

   @version 1.0 1/27/2017
```

```
   @param &value - The node we are adding to the back of the list

   @param key - ID key for list

   @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 );
  /*
   * YOU MUST IMPLEMENT THIS FUNCTION AS
   * PART OF THIS LAB
   * */
  if ( isEmpty() )
        firstPtr = lastPtr = newPtr;
  else
  {
        lastPtr->nextPtr = newPtr;
        lastPtr = newPtr;
  }

} // end function insertAtBack
/**

   Purpose:

      Remove a node from the front of the list
```

```
   @author Ron Sha

   @version 1.0 1/27/2017




   @param &value - node we are removing

   @return - none

*/
// Delete a node from the front of the list
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; // data being removed

      delete tempPtr;
      return true; // delete successful
   } // end else
```

```
} // end function removeFromFront
/**

   Purpose:

      Remove node from the back of the list

   @author Ron Sha

   @editted by Kenneth Surban

   @version 1.0 5/11/2017



   @param &value - desired node to remove from the back of the list

   @return - none

*/
// delete a node from the back of the list
template< typename NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{

            /*
             * Implement this function.  Use removeFromFront as an
             * example in implemeting this function
             * */
```

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

   if ( firstPtr == lastPtr )
      firstPtr = lastPtr = 0;
   else
   {

      ListNode< NODETYPE > *currentPtr = firstPtr;

         /*  this is where you need add more code
          * First, you need to start from the beginning of the linked list,
          * and traverse until the node before the last node.
          *
          * Once you got to the node before the last node, you need to
          * point the node before last node to the last node so you can
          * remove the last node
          *
          * */
                  while(currentPtr -> nextPtr != lastPtr){
                           currentPtr = currentPtr -> nextPtr;
 }
                  lastPtr = currentPtr;
                  currentPtr->nextPtr = NULL;

   // your codes here
   // more codes
```

```
      } // end else

      value = tempPtr->data;
      delete tempPtr;
      return true; // delete successful
   } // end else
} // end function removeFromBack
/**

   Purpose:

      Assign and return firstPtr to NULL if list is empty.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param - none

   @return - none

*/
// Is the List empty?
template< typename NODETYPE >
bool List< NODETYPE >::isEmpty() const
{
   return firstPtr == 0;
} // end function isEmpty
/**
```

```
   Purpose:

      Get new nodes by returning a pointer a new node.

   @author Ron Sha

   @version 1.0 1/27/2017




   @param &value - node to retrieve

   @return - ptr

*/
// Return a pointer to a newly allocated node
template< typename NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode(
   const NODETYPE &value, int)
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   return ptr;
} // end function getNewNode
/**

   Purpose:

      Print out if the list is empty and if not print out its elemenets.

   @author Ron Sha
```

```
   @version 1.0 1/27/2017



   @param - none

   @return - none

*/
// Display the contents of the List
template< typename NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) // empty list
   {
      cout << "The list is empty\n\n";
      return;
   } // end if

   ListNode< NODETYPE > *currentPtr = firstPtr;

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

   while ( currentPtr != 0 ) // display elements in list
   {
      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())
      {
       // data value is a simple data type and can be printed
       cout << currentPtr->data << ' ';
      }
      else {
        cout <<"Can't print - Not a simple data type (int, string, char, double)\n";
      }
     currentPtr = currentPtr->nextPtr;
   } // end while

   cout << "\n\n";
} // end function print

/**

   Purpose:

      Get info on list if it's empty and return NULL. If it's not point to
      the values in data.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param myKey - ID key required to getInfo
```

```
   @return - NULL(if list is empty) otherwise return values of data

*/
// 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: \n";

   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
} // end function print


/**
```

```
   Purpose:

      Print out the nodes in the list.

   @author Ron Sha

   @version 1.0 1/27/2017



   @param List< NODETYPE > & nodeList - list of nodes

   @return - none

*/
template< typename NODETYPE >
void printNoteInfo (  List< NODETYPE > & nodeList)
{
   NODETYPE *wp;
   wp = (NODETYPE *) nodeList.getInfo(0); //get node based on key

   ListNode< NODETYPE > *currentPtr;
   currentPtr =  nodeList.getFirstPtr();



   //cout << "\n 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
      currentPtr = currentPtr->getNextPtr();
      wp->printInfo();
   } // end while
}

#endif

#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

#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <iomanip>
#include "dbconnect.h"

/**

  Purpose:

    Set up connection with mySQL and print if there's an error.

  @author Ron Sha

  @version 1.0 1/27/2017



  @param connection_details mysql_details - mySQL login information

  @return - connection

*/
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;
}
/**

   Purpose:

      Use mySQL statements to print and sort database

   @author Ron Sha

   @version 1.0 1/27/2017



   @param *connection - Connect to mySQL

   @param *sql_query - mySQL statement

   @return - none

*/
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);
}
 // main.cpp
#include <iostream>
#include <typeinfo>
#include "LinkedList.hpp"
#include "person.h"
#include <mysql.h>
#include <stdio.h>
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <string.h>
#include <sstream>
#include "dbconnect.h"
#include "wine.h"
#include "printMeFirst.h"



using namespace std;
```

```
int main()
{
  printMeFirst("Kenneth Surban", "CS-116 - 2017 Spring");
  // mySQL implementation
  MYSQL *conn;    // the connection
  MYSQL_RES *res; // the results
  MYSQL_ROW row;  // the results row (line by line)

  // Wine LinkedList
  Wine w;
  List< Wine > wineList;

  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 *)"wine"; // the databse to pick

  // connect to the mysql database
  conn = mysql_connection_setup(mysqlD);

  // assign the results return to the MYSQL_RES pointer

    // use wine database
    res = mysql_perform_query(conn, (char *)"use wine");

  // console program
  int choice;
  int choiceOfOne;
  bool displayMenu = true;
```

```
//option select mySQL
double xa, xb; // price 1 and price 2
int ya, yb; // year 1 and year 2
int sa, sb; // score 1 and score 2
int sum;
int avg = 0; // average price
int z = 0;
char* sqlcmd;
string s;
ostringstream oss;

// menu selector
while (displayMenu != false ) {
  cout << "\n Welcome to the Wine Finder: \n";
  cout << " 1 - Find Wines by Score and Price \n";
  cout << " 2 - Find Wines by Price \n";
  cout << " 3 - Find Wines by Vintage \n";
  cout << " 4 - Exit \n ";
  cout << "Please enter your desired selection: ";
  cin >> choice;


  switch (choice)
  {
    case 1:
    {
      cout << "Selecting from the following \n";
      cout << " 1 - Display All Wines by Score then Price \n";
      cout << " 2 - Display All Wines by Score then Price, except the Last one \n";
      cin >> choiceOfOne;
      switch (choiceOfOne)
```

```
{
 case 1:
 {
  cout << "Enter Desired Rating\n";
  cin >> sa >> sb;
  oss << "select name, vintage, score, price, type from wineInfo where score between " << sa << " and " << sb
<< " order by price";
  s = oss.str();
  sqlcmd = (char *)s.c_str();
  res = mysql_perform_query(conn,sqlcmd);
  cout << left << setw(30) <<"Wine Name" <<
            left << setw(15) << "Vintage" <<
     left << setw(15) << "Rating" <<
     left << setw(15) << "Price"  <<
     left << setw(15) << "Type"
  << endl;

  row = mysql_fetch_row(res);

  while ((row = mysql_fetch_row(res)) !=NULL)
  {
  // convert (wineName) char * to string
  std::string wineName(row[0]);

  // convert char * to int
  std::string sWY(row[1]);
  stringstream streamYear;
  int wineYear;
  streamYear.str(sWY);
  streamYear >> wineYear;

  std::string sWR(row[2]);
```

```
stringstream streamRating;
int wineRating;
streamRating.str(sWR);
streamRating >> wineRating;

// convert char * to double
std::string wineType(row[4]);

// convert (wineType) char * to string
istringstream sWP(row[3]);
double winePrice;
sWP >> winePrice;

w.setInfo(wineName, wineYear, wineRating, winePrice, wineType);
wineList.insertAtBack(w,z);
sum = sum + winePrice;
z++;
}

printNoteInfo(wineList);

cout << "\nNumber of wines: " << z << endl;
avg = sum/z;
cout << "Average Price of wines: " << avg << endl;

// remove the elements in wineList
res = mysql_perform_query(conn,sqlcmd);
row = mysql_fetch_row(res);

while ((row = mysql_fetch_row(res)) !=NULL)
{
```

```
  wineList.removeFromBack(w);
  }

  oss.str("");
  /* clean up the database result set */
  mysql_free_result(res);
  break;
 }
 case 2:
 {
  cout << "Enter Desired Rating\n";
  cin >> sa >> sb;
  oss << "select name, vintage, score, price, type from wineInfo where score between " << sa << " and " << sb
<< " order by price";
  s = oss.str();
  sqlcmd = (char *)s.c_str();
  res = mysql_perform_query(conn,sqlcmd);
  cout << left << setw(30) <<"Wine Name" <<
          left << setw(15) << "Vintage" <<
     left << setw(15) << "Rating" <<
     left << setw(15) << "Price"  <<
     left << setw(15) << "Type"
  << endl;

  row = mysql_fetch_row(res);
  int z = 0;
  while ((row = mysql_fetch_row(res)) !=NULL)
  {
  // convert (wineName) char * to string s
  std::string wineName(row[0]);

  // convert char * to int
```

```
std::string sWY(row[1]);
stringstream streamYear;
int wineYear;
streamYear.str(sWY);
streamYear >> wineYear;

std::string sWR(row[2]);
stringstream streamRating;
int wineRating;
streamRating.str(sWR);
streamRating >> wineRating;

// convert char * to double
std::string wineType(row[4]);

// convert (wineType) char * to string
istringstream sWP(row[3]);
double winePrice;
sWP >> winePrice;

w.setInfo(wineName, wineYear, wineRating, winePrice, wineType);
wineList.insertAtBack(w,z);
sum = sum + winePrice;
z++;
}

for(int i = 0; i < 1; i++)
{
  wineList.removeFromBack(w);
  z = z - 1;
}
```

```
        printNoteInfo(wineList);

        cout << "Number of wines: " << z << endl;
        avg = sum/z;
        cout << "Average Price of wines: " << avg << endl;

        // remove the elements in wineList
        res = mysql_perform_query(conn,sqlcmd);
        row = mysql_fetch_row(res);

        while ((row = mysql_fetch_row(res)) !=NULL)
        {
        wineList.removeFromBack(w);
        }

        oss.str("");

        /* clean up the database result set */
        mysql_free_result(res);
        break;
      }
     }
     break;
    }
    case 2:
    {
     cout << "Enter Desired Price Range\n";
     cin >> xa >> xb;
      oss << "select name, vintage, score, price, type from wineInfo where price between " << xa << " and " << xb <<
" order by price";
      s = oss.str();
```

```
sqlcmd = (char *)s.c_str();
res = mysql_perform_query(conn,sqlcmd);

cout << left << setw(30) <<"Wine Name" <<
                left << setw(15) << "Vintage" <<
        left << setw(15) << "Rating" <<
        left << setw(15) << "Price"  <<
        left << setw(15) << "Type"
    << endl;



    while ((row = mysql_fetch_row(res)) !=NULL)
    {
     cout << setw(32) << left << row[0] << setfill(' ') // coulumn (field) #1 - Wine Name
      << setw(15) << row[1] << setfill(' ') // field #2 - Vintage
      << setw(15) << row[2] << setfill(' ') // field #3 - Rating
      << setw(13) << row[3] << setfill(' ') // field #4 - Price
      << setw(10) << row[4] << setfill(' ') // field #5 - Wine type
      << endl;

      istringstream sWP(row[3]);
      double winePrice;
      sWP >> winePrice;

      sum = sum + winePrice;
      z++;
    }

    cout << "\nNumber of wines: " << z << endl;
    avg = sum/z;
    cout << "Average Price of wines: " << avg << endl;
```

```
        /* clean up the database result set */
        oss.str("");

      mysql_free_result(res);
        /* clean up the database link */
      break;
    }
    case 3:
    {
      cout << "Enter Year\n";
      cin >> ya >> yb;
      oss << "select name, vintage, score, price, type from wineInfo where vintage between " << ya << " and " << yb
<< " order by vintage desc, score desc";
      s = oss.str();
      sqlcmd = (char *)s.c_str();
      res = mysql_perform_query(conn,sqlcmd);

      cout << left << setw(30) <<"Wine Name" <<
                        left << setw(15) << "Vintage" <<
            left << setw(15) << "Rating" <<
            left << setw(15) << "Price"  <<
            left << setw(15) << "Type"
        << endl;


        while ((row = mysql_fetch_row(res)) !=NULL)
        {
         cout << setw(32) << left << row[0] << setfill(' ') // coulumn (field) #1 - Wine Name
          << setw(15) << row[1] << setfill(' ') // field #2 - Vintage
          << setw(15) << row[2] << setfill(' ') // field #3 - Rating
          << setw(13) << row[3] << setfill(' ') // field #4 - Price
          << setw(10) << row[4] << setfill(' ') // field #5 - Wine type
```

```
            << endl;

            istringstream sWP(row[3]);
            double winePrice;
            sWP >> winePrice;

            sum = sum + winePrice;
            z++;
           }
           cout << "\nNumber of wines: " << z << endl;
           avg = sum/z;
           cout << "Average Price of wines: " << avg << endl;
           /* clean up the database result set */
           oss.str("");

        mysql_free_result(res);
           /* clean up the database link */
        break;
       }
      case 4:
      {
        mysql_close(conn);
        cout << "You are now exiting... \n";
        displayMenu = false;
        break;
      }
     } // end Switch

   } // end MenuDisplay

} // end main
```

Test Cases:

```
cs:LinkedList$ ./main

 Welcome to the Wine Finder:
 1 - Find Wines by Score and Price
 2 - Find Wines by Price
 3 - Find Wines by Vintage
 4 - Exit
 Please enter your desired selection: 1
Selecting from the following
 1 - Display All Wines by Score then Price
 2 - Display All Wines by Score then Price, except the Last one
1
Enter Desired Rating
70
92
Wine Name                     Vintage        Rating         Price          Type
Grgich Chardonnay               2013           90             43          White
Stags Leap Artemis Cabernet     2013           92             65          Red
Alpha Omega Chardonnay          2012           92             69.99       White
Silver Oak Cabernet             2011           91             110         Red

Number of wines: 4
Average Price of wines: 71

 Welcome to the Wine Finder:
 1 - Find Wines by Score and Price
 2 - Find Wines by Price
 3 - Find Wines by Vintage
 4 - Exit
 Please enter your desired selection:
```

```
 Welcome to the Wine Finder:
 1 - Find Wines by Score and Price
 2 - Find Wines by Price
 3 - Find Wines by Vintage
 4 - Exit
 Please enter your desired selection: 1
Selecting from the following
 1 - Display All Wines by Score then Price
 2 - Display All Wines by Score then Price, except the Last one
1
Enter Desired Rating
30
100
Wine Name                    Vintage        Rating         Price          Type
Grgich Chardonnay              2013           90             43           White
Stags Leap Artemis Cabernet    2013           92             65           Red
Alpha Omega Chardonnay         2012           92             69.99        White
Duckhorn Cabernet              2013           93             72           Red
Silver Oak Cabernet            2011           91             110          Red
Joseph Phelps Insignia         2013           97             240          Red
Opus One Bordeaux              2012           97             399.99       Red

Number of wines: 7
Average Price of wines: 142
```

```
cs:LinkedList$ ./main
 Program written by: Kenneth Surban
 Course info: CS-116 - 2017 Spring
 Date: Thu May 11 21:08:29 2017


 Welcome to the Wine Finder:
 1 - Find Wines by Score and Price
 2 - Find Wines by Price
 3 - Find Wines by Vintage
 4 - Exit
 Please enter your desired selection: 1
Selecting from the following
 1 - Display All Wines by Score then Price
 2 - Display All Wines by Score then Price, except the Last one
1
Enter Desired Rating
92
98
Wine Name                    Vintage        Rating         Price          Type
Alpha Omega Chardonnay         2012           92             69.99        White
Duckhorn Cabernet              2013           93             72           Red
Joseph Phelps Insignia         2013           97             240          Red
Opus One Bordeaux              2012           97             399.99       Red

Number of wines: 4
Average Price of wines: 195
```

```
 Welcome to the Wine Finder:
 1 - Find Wines by Score and Price
 2 - Find Wines by Price
 3 - Find Wines by Vintage
 4 - Exit
 Please enter your desired selection: 3
Enter Year
2011
2012
Wine Name                     Vintage        Rating         Price          Type
Opus One Bordeaux               2012           97             399.99       Red
Alpha Omega Chardonnay          2012           92             69.99        White
Silver Oak Cabernet             2011           91             110          Red

Number of wines: 3
Average Price of wines: 192
```

```
Number of wines: 6
Average Price of wines: 148
```

Note: Specifically the Silver Oak Cabernet rating was off in some. I ran the same statement in mySQL(very strange)…

```
mysql> select name, vintage, score, price, type from wineInfo where score between 10 and 100 or
der by score asc, price;
+-----------------------------+---------+-------+--------+-------+
| name                        | vintage | score | price  | type  |
+-----------------------------+---------+-------+--------+-------+
| Stags Leap Chardonnay       |    2014 |    90 |     30 | White |
| Grgich Chardonnay           |    2013 |    90 |     43 | White |
| Silver Oak Cabernet         |    2011 |    91 |    110 | Red   |
| Stags Leap Artemis Cabernet |    2013 |    92 |     65 | Red   |
| Alpha Omega Chardonnay      |    2012 |    92 |  69.99 | White |
| Duckhorn Cabernet           |    2013 |    93 |     72 | Red   |
| Joseph Phelps Insignia      |    2013 |    97 |    240 | Red   |
| Opus One Bordeaux           |    2012 |    97 | 399.99 | Red   |
+-----------------------------+---------+-------+--------+-------+
8 rows in set (0.00 sec)
```