```
 /*
    This is an assignment I did for my data structures class where I created a simplified version of an online retail system using the C++ STL
  */
#include "mydatastore.h" #include "user.h" #include "util.h" #include <iomanip> #include <list>
#include <map> #include <set> #include <sstream> #include <string> #include <vector>
using namespace std;
MyDataStore::MyDataStore ()
{
}

MyDataStore::~MyDataStore ()
{
// delete products and users to prevent memory leak for (unsigned int i = 0; i < products.size(); i++) {
  delete products[i];
}

for (unsigned int i = 0; i < user.size (); i++)
  {
    delete user[i];
  }}

void
MyDataStore::addProduct (Product * p)
{    // storing keywords for products in set set<string> keys = p->keywords(); set<string>::iterator itr; set<Product*> emptyProd;
// if a keyword doesn't have a set of products yet, then we add one
  for (itr = keys.begin (); itr != keys.end (); itr++)
    {
      if (keyword.find (*itr) == keyword.end ())
 {
   keyword.insert (make_pair (*itr, emptyProd));
 }
// adding products to keyword set
      keyword[*itr].insert (p);
    }
// keeping track of products being added
  products.push_back (p);
}

void
MyDataStore::addUser (User * u)
{
// creating map of usernames to user users.insert(make_pair(convToLower(u->getName()), u)); // keeping track of users being added user.push_back(u);
}

std::vector < Product * >MyDataStore::search (std::vector < std::string >
           &terms, int type)
{
  vector < string >::iterator it;
  set < Product * >keys = keyword[terms[0]];
// iterating through the terms
  for (it = terms.begin (); it != terms.end (); ++it)
    {    // finding the term in keyword map
      if (keyword.find (*it) != keyword.end ())
 {
// doing set intersection if the type is AND
   if (type == 0)
     {
       keys = setIntersection (keys, keyword[*it]);
// doing set union if the type is OR
     }
   else if (type == 1)
     {
       keys = setUnion (keys, keyword[*it]);
     }
 }
    }
// converting set to vector

  vector < Product * >finished (keys.begin (), keys.end ()); // copying finished vector to private class variable matches = finished;
  return matches;
}

void
MyDataStore::addToCart (string name, Product * p)
{
// converting username to lower and checking if user is valid name = convToLower(name);
  if (users.find (name) == users.end ())
    {
      cout << "Invalid request" << endl;
      return;
    }
// checking if user has a cart, if not then adding one
  if (userCart.find (name) == userCart.end ())
    {
      vector < Product * >product;
      userCart.insert (make_pair (name, product));
    }
// pushing back products into the cart
  userCart[name].push_back (p);
}

void
MyDataStore::buyCart (string user)
{
// converting username to lower, and checkng if user has a cart user = convToLower(user);
  if (userCart.find (user) == userCart.end ())
    {
      cout << "Invalid username" << endl;
      return;
```

```
    }
  vector < Product * >::iterator it;
// iterating through the cart
  for (it = userCart[user].begin (); it < userCart[user].end ();)
    {
// checking if the user is able to buy the product
      if ((users[user]->getBalance () - (*it)->getPrice () > 0)
   && ((*it)->getQty () > 0))
 {   // deducting the amount and subtracting quantity users[user]->deductAmount((*it)->getPrice());
   (*it)->subtractQty (1);
// erasing the item from user
   userCart[user].erase (it);
 }
      else
 {
// incrementing the for loop only if we didn't erase
   ++it;
 }
    }
}

void
MyDataStore::viewCart (string user)
{
// converting username to lower, and checkng if user has a cart user = convToLower(user);
  if (userCart.find (user) == userCart.end ())
    {
      cout << "Invalid username" << endl;
      return;
    }
  vector < Product * >::iterator it;
  int i = 1;
// iterating through user car and printing out the products and what item they are for (it = userCart[user].begin(); it != userCart[user].end(); ++it) {
  cout << "Item " << i << endl;
  cout << (*it)->displayString () << endl;
  i++;
}}

void
MyDataStore::dump (std::ostream & ofile)
{
// iterating through the products and printing them out onto the file name ofile << "<products>" << endl;
  vector < Product * >::iterator itr;
  for (itr = products.begin (); itr != products.end (); ++itr)
    {
      (*itr)->dump (ofile);
    }
  ofile << "</products>" << endl;
// iterating through the users and printing them out onto the file name

  ofile << "<users>" << endl;
  map < string, User * >::iterator itr2;
  for (itr2 = users.begin (); itr2 != users.end (); itr2++)
    {
      (*itr2).second->dump (ofile);
    }
  ofile << "</users>" << endl;
}
```