COMP322: Assignment 4- Winter 2012

Due at 11:30pm EST, 16th April 2012

1 Introduction

For this assignment, we will combine the recommenders we wrote in assignment two and three using inheritance. We also will get to create our OWN iterator and see how it can be used.

2 Getting Started

On the course website, you should download the file `SupportCodeA4.zip`. Inside here are the header files for the classes you will implement. In addition to the header files, there is a file `RatingInformation.cc` which contains the same definition using a struct as in assignment two and three. Finally there are a couple of very small txt sample files you can use as examples of the files you'll be reading from during the assignment. Note: There are potentially some issues related to spacing on windows based file systems vs linux based file systems. The ratings file should have 3 columns per line, the movie list file should have one movie per line, and the reviews file (the new one) should have first a number, then a tab and then some text on each line.

You will be writing the .cc files that should accompany these .h files. You will also write a program with a main function inside of it.

You are free and encouraged to add private methods/functions to the .h files but you must not delete or change the prototypes of the existing ones.

For full credit on this assignment, you must not include any source code files (other than RatingInformation.cc) inside any other source files. You may include header files. This means that to compile the assignment you should write (all on one line)

```
g++ MovieInformation.cc RatingInformation.cc
            SimpleRecommender.cc RecommendationProgram.cc
              Recommender.cc ReviewCollection.cc ReviewCollectionIterator.cc
                CollaborativeRecommender.cc
```

OR

```
g++ *.cc
```

Note that this is where if we had more time, we would most certainly go into the idea of makefiles as a way to simplify the compilation process.

This assignment will have two main parts to it:

1. First we will move some of the code that was previously in SimpleRecommender.cc into an abstract class defined in Recommender.cc . We will then extend the collaborative filtering code (where we provided a rating based on common neighbors) from assignment two so that it too can provide recommendations. You can take your own answers from

the previous assignments or you are welcome to copy code as desired from the solutions for assignments two and three for this. (Assignment three solutions will be posted early in the week of March 26th)

2. In the second step, we will define a new type that is meant to maintain a collection of Reviews, both positive and negative. In order to make this type useful with the algorithms in the standard library, we will define a second type `ReviewCollectionIterator` which allows us to traverse the two sets of reviews in a specific way.

Here we describe what each class should do. Note that in general, you will be implementing several .h files. You may always add protected or private helper methods or member variables as needed to them.

3 Using Inheritance

3.1 Recommender.cc

In this file, you should complete the source code for the header file `Recommender.h`. Notice that the header file contains a virtual function `generatePrediction` set to be 0 but that `makeRecommendations` is not virtual. Recall that setting a function equal to 0 this way is how we define an abstract method in C++. This means that we can not create an instance of a `Recommender` directly.

`makeRecommendations` on the other hand is not virtual or set to be 0. The idea here is that regardless of what recommendation technique we use to generate a prediction for a movie, the overall selection at the end will still be the same of choosing the top n movies from the list. (In theory, this method could be virtual as well but not set to be 0. This would be useful if we wanted to use a different scheme for choosing the top n movies. For example, choosing the top movie from every genre. In practice, it's better to keep the method non-virtual for now as it's very easy to add virtual later if we want.)

Completing this class should be relatively simple. You will need to copy necessary methods/classes/functions from the old `SimpleRecommender.cc` from assignment three to this source file.

Notice also that `ratings` and `movies` are *protected*. If we did not make them protected it would be impossible to access them from our child classes, which would make it difficult to make a prediction.

3.2 SimpleRecommender.cc

This file should be almost identical to the one in assignment three except for two major changes:

1. The methods other than `generatePrediction` (and any associated helper methods/functions/classes) should be removed since they are now part of the `Recommender` class.

2. Inside the header for the class in the .h file, we have added `: public Recommender` which means `SimpleRecommender` extends `Recommender`

3.3 CollaborativeRecommender.cc

In this class you should define a different recommendation method using an approach similar to what we wrote in assignment two. First define a class `CollaborativeRecommender` which extends `Recommender`

In assignment two, we wrote a method `computeSimilarity` which computed the similarity between two users. We will now use that to generate a prediction.

Write a method `generatePrediction`. This method should have an identical header to that in `SimpleRecommender.cc` or the abstract header in `Recommender.cc`. This method takes as input a `userid` and a `movieName`. It should do the following:

1. First make a list of all the users that have rated `movieName`.
2. If none of the users rated that movie, you should return `5` (the average middle rating)
3. Assuming there exists another user who rated this movie, you should find the user who is most similar to the user with id `userid` using the method you wrote in assignment two. (You can use as much code from the solutions as you like. You'll have to modify things like the headers slightly.)
4. Your method should return the same rating for the `movieName` as the user who is most similar rated it.

3.4 RecommendationProgram.cc additions (part one)

Now that you have defined things this way, you should be able to make the following addition to your `RecommendationProgram.cc` file.

Create a pointer to a `Recommender` object. Before asking the user to make recommendations, ask the user which kind of recommender he/she would like to use. If the user types "SimpleRecommender" then you should assign to the pointer a new `SimpleRecommender` object. If the user types "CollaborativeRecommender" you should assign to the pointer a new `CollaborativeRecommender` object. In either case, you should have the exact same code after this as you should *NOT* need to keep track anywhere of which recommender you are using after the initial if statement.

4 Creating a ReviewCollection and iterator

We will now define a new type `ReviewCollection` along with a `ReviewCollectionIterator`. The idea of defining these in conjunction will be that by defining an iterator to our collection as well, we can use any of the algorithms or other techniques that we've been using up until now. The main purpose of this exercise is to see how to define an iterator ourselves.

4.1 ReviewCollection.cc

You should define a class `ReviewCollection` for which the header file is provided in `ReviewCollection.h`. The `ReviewCollection` class itself is not very complex. (Most of the work is

in the `ReviewCollectionIterator` class.) A `ReviewCollection` consists of two `vector<string>`, one of which stores the positive reviews for a movie and another of which stores the negative reviews for a movie. A `ReviewCollection` also contains a movie id storing what movie the collection is of.

The class has 8 methods along with a constructor:

1. The constructor takes as input a string filename and int id. The id represents the movieid for this `ReviewCollection`. The `string filename` stores a list of the reviews to load. Each line of `filename` will contains first a number then a tab and then some text. The text is the review itself. If the number is greater than or equal to 0, then the text should be added to a positive review. If the number is less than 0, then the text should be added to a negative review. See reviews.txt for an example file.

   If the file is not found, then your constructor should leave the two lists empty. It is important for the main function we write later to make sure your program does *not* crash in this case.

2. `begin()` This method returns a start iterator to iterate over this collection. The type of the iterator is the type that you will be defining in the next part.

3. `end()` This method returns an end iterator which will denote the end of the collection. To figure out the exact thing to return in `begin` and `end` you will want to think about the constructor for `ReviewCollectionIterator` as well as what `==` means. (Remember that one will use the iterator by saying something like

```
for (ReviewCollectionIterator  current = collection.begin();
         current != collection.end(); current++)
```

4. `positiveIterStar()` This returns an iterator to the start of the positive review container. (Just return the result of calling the method defined on `vector`)

5. `positiveIterEnd()`, `negativeIterStart()`, `negativeIterEnd()` are defined similarly to be iterators to the two different review collections. (These are all 1 line methods)

6. `getId()`: This is a getter to allow you to get the movie id for this collection.

7. `size()` : This method returns the number of reviews total (positive plus negative).

4.2 ReviewCollectionIterator.cc

Here you will define your iterator that iterates over the `ReviewCollection`. The goal of defining this iterator will be to alternate the ratings between positive and negative. We will first show a positive review, then a negative, then a positive, etc. until one list runs out at which point we will simply show all of the rest of these.

A `ReviewCollectionIterator` will thus need to store 5 things:

1. An iterator pointed at the current position of the positive review vector

2. An iterator pointed at the current position of the negative review vector
3. An iterator pointed one past the end of the positive review vector
4. An iterator pointed one past the end of the negative review vector
5. A bool `positive` storing whether the `ReviewCollectionIterator` is currently pointing at a positive or negative review.

To make this work, you need to overload several operators. Remember that the headers for each operator being overloaded will be the same as in the .h file except with an additional `ReviewCollectionIterator::` after the return type.

The operators you need to overload are:

1. `=` operator to allow one to assign an iterator to another (e.g. ReviewCollectionIterator current = reviews.begin())
2. `==` operator to allow a comparison (e.g. if (current == reviews.end()))
3. `!=` operator to allow negative comparisons (e.g. (current != reviews.end())
4. `++` operator (prefix and postfix) to allow forward traversal of the collection (e.g. current++) This method is the trickiest to implement. Based on whether `positive` is true or not, you'll either want to increment the positive iterator or the negative iterator. You will then want to change `positive` to be the opposite UNLESS the other iterator is completed. You know it's completed if the other iterators start and finish are the same. For example: If `positive` is true, and `negativeStart` and `negativeEnd` are NOT the same, then you would increment `positiveStart` and change `positive` to be false. However, if `negativeStart` and `negativeEnd` are equal, then there are no more negative reviews left because this iterator has reached the end. In this case you'd just increment `positiveStart`. Once both start iterators equal the end iterators, then your `==` method should match with the end of the iterator of the collection if you had a command such as `current == collection.end()`
5. `*` operator to get the value of the current iterator. The value in this case should be a string of the particular review data.

Hint: Remember that if you have a `ReviewCollection collection`, the way someone will iterate will be something like:

```
for (ReviewCollectionIterator current = collection.begin();
         current != collection.end(); current++)
```

It is important to make sure that `current` and `collection.end()` eventually reach the exact same point. This means you will need to be careful to make sure that ALL properties of `collection.end()` and `current` are the same, including the `isPositive` property. If you are not careful, you could reach the end of your collection and have all properties be the same except for the `isPositive` property.

There are a few ways to address this: Inside your `++` method, you could make sure that the iterator always ends with a consistant value for `isPositive`. Alternatively, inside your `end()` method, you could do a calculation to determine whether the iterator will end with `isPositive` true or false and return an appropriate `ReviewCollectionIterator` based on that.

Updated clarification: You also, as suggested in the header file, should define a constructor. The constructor should take 5 things as input: the 5 properties that define your iterator. It should simply initialize the 5 properties to be equal to the 5 values passed in as input.

We'll talk some about defining our own custom iterator types in class, but a very useful tutorial for doing this is at `http://www.oreillynet.com/pub/a/network/2005/11/21/what-is-iterator-in-c-plus-plus-part2.html?page=3#heading3` (pages 1 and 2 are worth reading but the key code examples are on pages 3 and 4)

5 RecommendationProgram.cc (part 2)

Now you should make a second set of changes to your `RecommendationProgram.cc`

1. Before your main loop, create a vector of `ReviewCollection`. Now, for each number i from 0 until 100, look for the file `reviews_i.txt`, generate a `ReviewCollection` based on that file, and add it to your vector. The idea is that the file `reviews_0.txt` will have all the reviews for the movie with id 0, and so you'll put it into the vector at position 0. Some of these files may not be found, in which case you'll have a `ReviewCollection` of size 0.

2. Now as before, ask the user the kind of recommender they wish to use and then enter a loop.

3. In your loop now, instead of always asking the user to enter a user id, first ask the user whether they'd like to get a recommendation or read reviews (you can use ints for this). If they choose recommendation, then do as before. If they choose reviews, then you should ask the user what movie id they'd like a review of, then ask them the kind of reviews they'd like (positive, negative, or mixed) and then, using your iterator, print all of the reviews to the screen in the alternating order described above. (For all of these, you may use numeric codes for simplicity rather than reading strings)

6 Submitting your assignment

What To Submit

`all .h and .cc files that you used for the assignment`
`readme.txt` (optional but strongly recommended) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more

partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.