CSS-340

03 November 2020

Pseudocode for files

For the App source file

Pseudocode:Default constructor

```
Apps::Apps(){   // Sets the default values for the private values in the Apps class
   name;
   category;
   rating;
   installs;
}
```

Pseudocode:Second Default constructor

```
Apps::Apps(string name, string category, double rating, float installs)
{
    Name;
    Category;
    Rating;
    Installs;
]
```

Pseudocode:Getters

```
string Apps::GetName() const{
    Returns name;
]
string Apps::GetCategory() const{
   returns category;
}
double Apps::GetRating() const{
   returns rating;
}
float Apps::GetInstalls() const{
   returns installs;
}
void Apps::PrintInfo() const {  //Prints the values of the app
   Print out the results and number of ratings
}
```

Pseudo for Ranking source file:

```
         RankingApp::RankingApp(Apps app)//reference .h file {
         Read app
         }
         Apps RankingApp::GetApp() const{
                  Return app;
}
void RankingApp::PrintInfo() const {
Declare two string values for reading
Declare  two ints one as a temp
```

Declare a data type of double for rating and installs;

Open your file/directory

Create a loop inside to check the file

Inside the file make a condition to retrieve the category, ranking, and # of installs

    The new if statement where the getter rating is equal to the that is requested then make a

Nested if statement to compare installations if true then displays rank and close file.

```
{
        if(app.getter==rating)
        if(app.category==installs)
                if(comparison of category and size )
                        move to a temp rank
                        close file
        }
Then close the file
Print the ranking info by this case by our limit of 500
Then display what rank it currently is.
}
```

Pseudocode Algorithms:

The sorting algorithm is bubble sort. Wanted to implement but had trouble doing so.

```
void bubbleSort(int arr[], int n)
{
        int i, j;
        for (i = 0; i < n-1; i++)
```

```
    // Last i elements are already in place
    for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
```

Built-in datatypes:

In our main we have 3 data types string, double, and int that get called back to be used in the other 2 classes. We have 2 classes, the App header contains the private data members string name, category, double rating, and long long installs. The Ranking header contains the data type int ranking. We had to use a long long variable because some of the install numbers for the apps on the text file went over the int number limit.

User-defined datatypes:

For our user defined datatypes, as I mentioned there are 2 classes both containing individual header files and a source file. In the App header file we have seven public functions. One default constructor and a second constructor to initialize the private fields. Then, we have string GetCategory() const;, double GetRating() const; float GetInstalls() const;void PrintInfo() const. The getters just retrieve the information, but the print function displays the name,rating, and number of installs. For the second header file, Ranking.h, contains RankingApp(Apps app) setting the app and the getter Apps GetApp() const to retrieve the information. Finally the last function in this header is the void PrintInfo() const and inside this function a file was opened making it so that the file could be read. Within that function it searches for the ranking and displays the result.

Summary and Contributions:

The goal of our program is to predict the rating of any app using the category it is based on and the use of the app in that category. In order to accomplish this, we defined classes Apps and RankingApp () and made header files for both classes. The Apps class takes the name and rating of the app, the category it comes from, and the number of downloads and returns the info for the RankingApp (). The RankingApp () then uses this info to see if the app is in the top 500 ranking of the Google Play Store and returns the ranking number. It does this by the reading the text file that has a list of the top 500 as of November 2, 2020. We assign the value on the list to variables and check if it matches with the user input if it does not it continues to go down the list until it finally all matches in which it sets the value of the apps rank. Afterwards we print it out for the user.

Some strategies we used to cut back on runtime we used was closing the text file after acquiring the rank since there is no reason for the program to continue through the list after we get the needed information. We also changed the method of comparing the categories in RankingApp () to the compare () function rather than a for loop to check if the first characters of the user input category match the text file's category. We also used different class files to reduce the run time and to keep everything orderly rather than having everything on the main class file.

Gabriel implemented most of the code which include the main class, the Apps () classes, RankingApp () classes and the mytext.txt file, also contributed to the summaries. Oscar made the outline for the project and came up with the format of the program, helped with debugging the code and wrote all the pseudocode for the project. Issac documented and summarized the coding for the written portion.