**Project 1 Specifications**

***Binary Search Tree***

Please review the general [*Program Guidelines and Evaluation Criteria*](https://aacc.instructure.com/courses/64613/pages/project-guidelines-and-evaluation-criteria) for information about projects, and download files needed for this project.

***Learning Objectives:***

1. Develop a high-quality C++ program, following specified guidelines and incorporating javadoc-style comments.
2. Draw a complete UML class diagram for the program.
3. Create a makefile for a C++ program.
4. Incorporate binary search tree data structure software written by the textbook author into your program.
5. Create a user-friendly menu-driven human/computer interface.
6. Input data from a text file.
7. Apply secure programming practices when using text files.
8. Create records for a binary search tree based on data from a text file.
9. Create reports from the data in a binary search tree.
10. Traverse a binary search tree.
11. Apply secure programming practices when using a binary search tree.
12. Use a binary search tree to solve a problem.
13. Create output that is accurate, complete, easily identifiable, and user-friendly.

***Specifications:***

Develop a high-quality, menu-driven object-oriented C++ program that creates a small database, using a binary search tree structure to store and process the data.  The database will contain the top 100 highest grossing films of 2015.

The C++ object-oriented program must use the BinarySearchTree class from the textbook (along with the BinaryNode, BinaryNodeTree, BinaryTreeInterface, NotFoundException, and PrecondViolatedExcep classes it requires) to fulfill the requirements of this project. When designing and implementing the program, apply good software engineering principles. Create a makefile for the program that allows compilation and execution of the program from within jGRASP using the Windows operating system. Start the analysis and design process by drawing a complete UML class diagram for the program that includes all the classes that are contained in the program, including the classes provided by the textbook and the classes that you create.  The UML class diagram will assist you in understanding the components of the project.

Your program must include:

* a **Film** class that stores all the data for a Film and provides appropriate methods to support good software engineering principles,
* a **FilmDatabase** class that stores the binary search tree and provides appropriate methods to support good software engineering principles,
* a **Menu** class that contains, as a minimum, a method for each menu used in the program that displays the menu and responds to all menu choices made by the user, calling appropriate methods in the FilmDatabase class as necessary, and
* a file named Project1.cpp that contains the main() method.

You may also include other classes, as needed. Note that the binary search tree must store Film objects, and the >, <, and == operators must be defined for that class because the BinarySearchTree class uses those overloaded operators.

The downloaded files include the Student class (Student.h and Student.cpp), the StudentDatabase class (StudentDatabase.h and StudentDatabase.cpp), BSTTest.cpp, and makefile for you to use as an example to help you begin your program. The program compiles, links, and executes if you **Make** it using jGRASP/Windows.

***Data Details:***

The database contains data pertaining to the 100 highest grossing films of 2015.  A comma delimited file named Films2015.csv contains the initial data. Each record is stored on one line of the file in the following format:

**Data                         Data type**

Rank int

Film Title (key) string

Studio string

Total Gross double

Total Theaters int

Opening Gross double

Opening Theaters int

Opening Date string

Each of the data fields is separated in the file using the comma (,) character as a delimiter. There is no comma (,) character after the last field on the line. The data is considered clean; There are no errors in the input file.

When storing the data in the binary search tree, use the data types shown above. **The Film Title will serve as the key field (its value is unique)**.

***Menu Details:***

Your program must begin by inputting the text data from the **Films2015.csv** text file and building a binary search tree for the Films, in order by the key (the film title). Then the program must display the following Main Menu:

MAIN MENU

D - Describe the Program

R – Reports

S - Search the Database

X - Exit the Program

Enter Selection ->

All menu choices are selected by typing the number of the choice.

**D - Describe the Program**

If the user chooses **Describe the Program**, the program provides a detailed description for the user, explaining what the program is doing and how it works. Note that this method does NOT substitute for javadoc-style comments. The audience for this method consists of non-technical users that have no information at all about the assignment. After providing the description, display the MAIN MENU again. DO NOT use recursion to do this; use a loop.

**R - Reports**

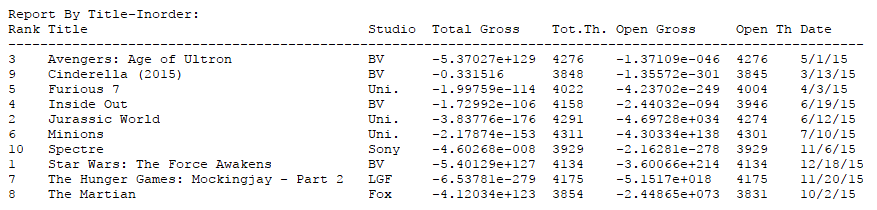
If the user chooses **Reports**from the MAIN MENU, the program displays the following menu:

REPORTS MENU  
                                                                           T - Order by Film Title report  
                                                                           R - Order by Rank report  
                                                                           X - Return to main menu

                                                                           Enter Selection ->

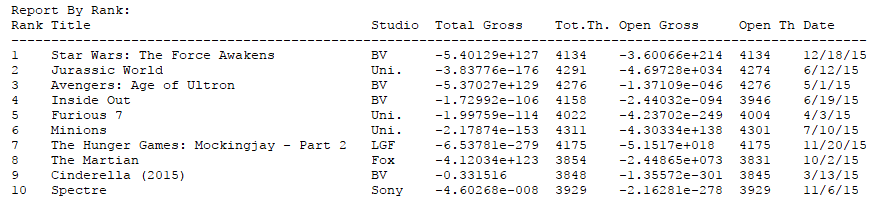
If the user chooses **Order by Film Title report** from the REPORTS MENU, the program should do the following:

1. Display a report in order by Film Title that contains all the data for each record stored in the binary search tree.
2. Identify the report and label all the data displayed appropriately. The following is a sample output (Assume there are only 10 nodes in the tree):



If the user chooses **Order by Rank report** from the REPORTS MENU, the program should do the following:

1. Display a report sorted in increasing order by earnings rank that contains all the data for each record stored in the binary search tree. Note that **this does not involve a simple tree traversal or single method call**. You will need to write code that specifically performs this function. You may assume that the rank is never less than 1 and never greater than 100. **DO NOT copy the data into another binary search tree, into an array, into a linked list, vector or into any other type of data structure**. Retrieve the data directly from the binary search tree which is stored in order by the Film Title.
2. Identify the report and label all the data displayed appropriately. The following is the sample output:



If the user chooses **Return to main menu** from the REPORTS MENU, then display the MAIN MENU again. Do NOT use recursion to do this; use a loop.

**S - Search the Database**

If the user chooses **Search the database**, the program should display another menu, as follows:

              Search Menu

T - Search by Title

K - Search by Keyword(s)

S - Search by Studio

M - Search by month of release

X - Return to main menu

Enter Selection ->

If the user chooses **Search by Title**, the program should do the following:

1. Request a film title from the user.
2. Search the database for the title given.  This will be an exact match of the film title and at most, only 1 record will match.  A **case insensitive search** should be performed.
3. If the film title is found, display the record stored in the binary search tree.  Otherwise report that the requested film title was not found.
4. Identify the output and label all the data displayed appropriately.

If the user chooses **Search by Keyword(s)**, the program should do the following:

1. Request title keywords from the user.
2. Search the database for all titles that contain the keyword(s). The keywords given represent all or a portion of the title.  When multiple keywords are given, only titles that contain the keywords in the order given will be selected.  For example, if the user enters ‘Star Wars’, “Star Wars: The Force Awakens” will be selected as will “The Making of Star Wars”.  However, “One Star, Many Wars”, “Once Upon a Star”, “Wars” and “Wars with Stars” will not be selected.
3. Multiple search keywords may be entered by separating the keywords with a comma.  All titles containing one or more of the keywords will be selected. For example, if the user enters 'Cinderella,SpongeBob,Dinosaur', "Cinderella", "The Good Dinosaur", "In Search of Dinosaurs", "The SpongeBob Movie: Sponge Out of Water" as well as "Cinderella Loves Dinosaurs and SpongeBob!" will be selected.
4. Display all records stored in the binary search tree where the title contains the keyword(s) given by the user.  If there are no records containing the keyword(s), report that the requested keyword(s) were not found.
5. Identify the output and label all the data displayed appropriately.

If the user chooses **Search by Studio**, the program should do the following:

1. Request the name of a Studio from the user.
2. Search the database for an exact match of the studio name.
3. Display all records stored in the binary search tree where the studio name matches the name given by the user.  If there are no records with the studio name given, report that the requested studio was not found.
4. Identify the output and label all the data displayed appropriately.

 If the user chooses **Search by month of release**, the program should do the following:

1. Request a month value between 1 and 12 from the user.
2. Search the database for all records where the release date contains the month requested. Note: the date is stored as a string.
3. Display all records stored in the binary search tree that match the month given by the user.  If there are no records with the requested release month, report that the requested month was not found.
4. Identify the output and label all the data displayed appropriately.

If the user chooses **Return to main menu**, then display the MAIN MENU again. Do NOT use recursion to do this; use a loop.

**Note:** To perform searches, you need to use binary search tree methods, such as contains(), getEntry() and so on.

**X – Exit the Program**

If the user chooses **Exit the Program** from the MAIN MENU, the program ends.

Since the program is controlled and manipulated by a human user, be sure to perform adequate error checking on every aspect of the program.

Be sure to follow the *Project Guidelines & Evaluation Criteria*, since the project will be evaluated using this criteria. In addition, be sure to use javadoc-style comments appropriately. When creating javadoc-style comments, keep in mind that the comment will eventually become part of an html file that will be used by other programmers on your programming team, and by maintenance programmers. Remember also that maintenance programmers have not seen the assignment (the specification), so the information you are providing here must provide all of the detailed information another programmer will need to completely understand the program and to maintain the code.

***Grading rubrics***

**Output (30 points)**

1. The makefile is correct
2. The program inputs all the data from the correct text file and correctly builds the binary search tree
3. The program displays a description that is adequate for a non-technical user who is totally unfamiliar with the program and/or the assignment
4. The program displays the menus correctly, accepting both uppercase and lowercase letters for the user's choice, responding correctly to each choice, and re-displaying the appropriate menu after processing the user's request
5. The program displays complete, correct, appropriately titled detailed reports as requested by the user
6. The program displays complete, correct, appropriately titled detailed search results as requested by the user

**Algorithm (50 points)**

1. Film class:
   1. Stores the specified data elements as private data members
   2. Contains methods that support good software engineering principles, support error handling and fail-safe programming, use appropriate c++ constructs, and support modularity
2. Filmdatabase class:
   1. Stores appropriate data elements as private data members
   2. Contains methods that support good software engineering principles, support error handling and fail-safe programming, use appropriate c++ constructs, and support modularity
3. Menu class:
   1. Stores appropriate data elements as private data members
   2. Contains methods that support good software engineering principles, support error handling and fail-safe programming, use appropriate c++ constructs, and support modularity
4. Project1.cpp file:
   1. Contains the main() method
   2. Supports good software engineering principles, supports error handling and fail-safe programming, uses appropriate c++ constructs, and supports modularity
5. Other requirements:
   1. If the program is unable to read the input file, the program provides an appropriate error message and behaves in a reasonable way

**Style (10 points)**

1. The program adheres to all the style criteria for documentation
2. The program adheres to all the style criteria for readability
3. The program adheres to all the style criteria for modifiability

Top of Form