Practical Design Document

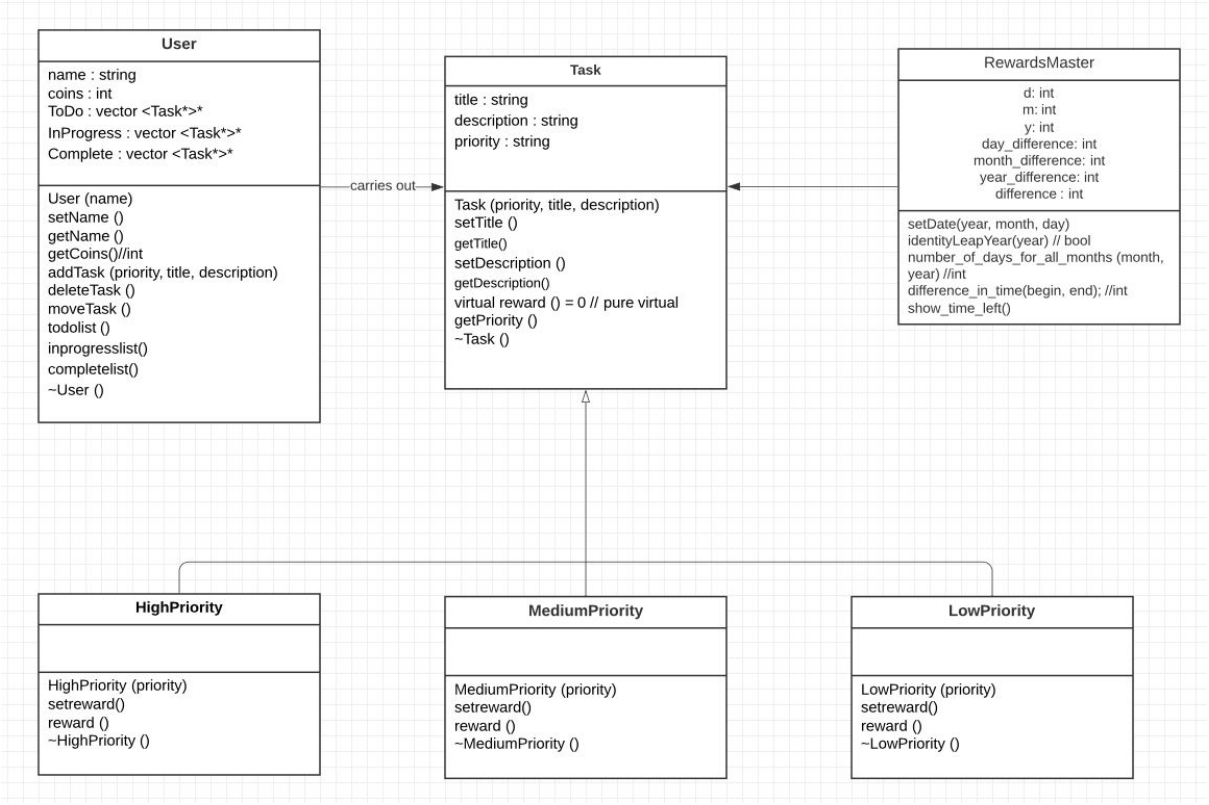
Specification of the Task

1. **Memory allocation from the stack and the heap**
    - **Arrays:** vectors used
    - **Strings:** states stored on stack
    - **Objects:** stored on heap, accessed with pointers.
2. **User Input and Output**
    - **I/O of different data types:** string input for name and task details, integer input for menu options.
3. **Object-oriented programming and design**
    - **Inheritance:** High, Medium and Low Priority subclasses inherit from Task.
    - **Polymorphism:** Each priority subclass has a different rewards system specific to the class.
    - **Abstract Classes:** Task is an abstract class with pure virtual function for reward.
4. **Testing**
    - **Test Inputs / Expected Outputs:**
    - **Unit Testing: Will be testing all important functions in each class to determine its functionality and ensures that it works as expected. The test result would be present when the user manually compile the programs or entered the command "make test"**

    - **Automated Testing:** Will test the program based on input and output testing, as well as testing for the difference between a text file with inputs and another test file with the expected output. We may need to use at least one input.txt and a output.txt that will be presented in the makefile in order to test the program

    - **Regression Testing:** Both valid and invalid inputs would be tested in the program using text files to test out the expected outputs. If an invalid input is entered, error messages are expected to be outputted and user may be allow to enter another input until the input is validated

Class diagram



Class descriptions

Variables:

User

- **name - stores the name entered by the user**
- **coins - stores the total amount of coins held by the user**
- **ToDo - stores a list of to do tasks entered by the user**
- **InProgress - stores a list of in progress tasks entered by the user**
- **Completed - stores a list of completed tasks entered by the user**

Task

- **title - stores the title of the task entered by the user**
- **description - stores the description of the task entered by the user**
- **checklist - stores the checklist of the task entered by the user**
- **notes - stores the notes of the task entered by the user**
- **priority - stores the priority of the task entered by the user**

RewardsMaster

- **bd, bm, by, ed, em , ey - stores the day, month and year respectively of the dates entered by the user**
- **day_difference, month_difference, year_difference - stores the difference in day, month and year respectively between the completion and due date of the task**

Methods:

User

- **setName, a setter function to allow the user to set the name they entered**
- **getName, a getter function that returns the name of the user**
- **addCoins, a function that adds the number of coins in the user's account based on the difference between the date of completion and the due date**
- **removeCoins, a function that removes coins when the user spent the coins on rewards**
- **getCoins, a getter that returns the total number of coins left by the user**
- **addTask, a function that has a Task as parameter, allows numerical input of 1,2 or 3 from the user and adds tasks to the destined list based on the number entered**
- **deleteTask, a function that removes a particular task based on the user's preference**
- **moveTask, a function that move the task to the assigned list**
- **todolist () - a function that lists out all tasks presented in the to do list**
- **inprogresslist() - a function that lists out all tasks presented in the in progress list**
- **completedlist () - a function that lists out all tasks presented in the completed list**

Task

- **setTitle() - a function that allow user to enter the title of the task**
- **setDescription() - a function that allow user to enter the description of the task**
- **setChecklist() - a function that allow user to enter the checklist of the task**
- **setNotes() - a function that allow user to enter the notes of the task**
- **reward() - a pure virtual function that displays the coins rewarded from the user**
- **getPriority() - a function that gets the priority of the user**

RewardsMaster

- **setDateTaskCompleted(int year, int month, int day) - a function that allows user to get the date of completion of the user's task**
- **getCompletedYear() , getCompletedMonth(), getCompletedDay() - getters that allows user to get the year, month and day of completion of the user's task**
- **setDateTaskDue(int year, int month, int day) - a function that allows user to get the due date of the user's task**
- **getDueYear() , getDueMonth(), getDueDay() - getters that allows user to get the year, month and day of the due date of the user's task**
- **identity_leap_year() - a function that identifies the leap year when the user entered a year in the parameter**
- **number_of_days_for_all_months() -a function that calculates the number of days based on the month and the year difference between the date of completion and the due date**
- **difference_in_time() - a function that calculates and returns the difference of days between the date of completion and the due date, which would be used in the calculation of rewards earned by the user.**

- **getRewardsList() - a function that lists out the rewards earned by the user in a vector<string>**

HighPriority

- **setReward() - a function that calculates the exact amount of coins to be rewarded based on the time.**

MediumPriority

- **setReward() - a function that calculates the exact amount of coins to be rewarded based on the time.**

LowPriority

- setReward() - **a function that calculates the exact amount of coins to be rewarded based on the time.**

Intended to use a While loop for main menu options.

Testing plan

- **Outline your testing strategy (1 mark)**

First we will have unit testing for each class object to ensure every state and behaviour is working correctly. This would be tested on separate cpp files to test whether each function in the class got the expected outputs. It would be named as <class name>Tester and is expected to run when the call “make test is run”. For the unit testings, we have conducted unit testings for the classes RewardsMaster, HighPriority, LowPriority and MediumPriority. We plan to conduct unit tests on User class before week 11 after finalising our main function design and ensure that the program works when tested.

We have also constructed a makefile for both automated and regression testing. This would ensure that the program will still perform as expected when there are any changes applied. Integration Testing, such as input and output testing, will be conducted using text files. When the command “make all” is entered, the program is expected to run the main function directly as well as conducting the unit tests once the user exits the program by entering the number “9”. The program is expected to pass all unit tests and the makefile should be able to compile by now using the command “make all” and “make test” for tests only. Integration testing using test files are expected to be completed at most by this weekend to make room of adding more new features or make improvements for the program

RewardsMaster class :

Under the filename <RewardsMasterTester>
13 tests to demonstrate the usability of each behaviour in the program

- **setDateTaskCompleted**()
    - Two test cases with different dates (2001/12/22 and 1970/04/08 respectively) are used to test on the program

    - A defined variable of the class type “RewardsMaster” is present to allocate the year, month and day entered by the user
    - The tests should pass if the parameter of the function, which are year, month and day stored in the variable is numerically identical to the parameters entered in the test
    - Otherwise, if the test fails, an error message “**setDateTaskCompleted** test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function
- **setDateTaskDue**()
    - Two test cases with different dates (2001/12/22 and 1970/04/08 respectively) are used to test on the program
    - A defined variable of the class type “RewardsMaster” is present to allocate the year, month and day entered by the user
    - The tests should pass if the states of the RewardsMaster class type, which are year, month and day stored in the variable is numerically identical to the parameters entered in the test
    - Otherwise, if the test fails, an error message “**setDateTaskDue** test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function
- identify_leap_year()
    - Two test cases with different years (2020 and 1903) are used to test on the program.
    - A defined variable of the class type “RewardsMaster” is present to call the function
    - The tests should pass if the boolean output of the function is 1 when the year is identified as a leap year,while the output would be 0 if the year is a non-leap year
    - Since 2020 is a leap year but 1903 is not a leap year, the expected output would be 1 and 0 respectively
    - Otherwise, if the test fails, an error message “identify_leap_year test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function
- number_of_days_for_all_months()
    - Four test cases with different years and months are used to test on the program.
    - A defined variable of the class type “RewardsMaster” is present to call the function
    - Different months that have either a leap year or a non-leap year are tested as the parameter of the function to calculate the total number of days in a particular month and year
    - When the second month is entered as 2 in the parameter, the output would be 29 if the year in the function’s parameter is identified as a leap year, and 28 if the year in the function’s parameter is identified as a non - leap year
    - The tests should pass if the total number of days calculated by the function based on a particular month and year is equivalent to the expected output (etc. 31 days on March 2020 and 30 days on November 2020)

        - Otherwise, if the test fails, an error message “number_of_days_for_all_months test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function
- difference_in_time()
    - Two test cases with different number of days difference are used to test on the program.
    - A defined variable of the class type “RewardsMaster” is present to call the function, while two setDate() functions with allocated dates are assigned to calculate the difference in days between two dates
    - This function takes in the dates of two different variables with class type <RewardsMaster> and calculates the exact days of difference between the two dates
    - Should the output be invalid (etc, months are more than 12 or less than 1, years are more than 10000 or less than 0), the program would request the user to reenter the invalid variable.
    - The error message “Invalid!” would be displayed in the compiler and stop the program entirely if the start date is more than the due date.
    - The first test uses the two equal dates as the parameters to test for the function when there are no day differences between the dates. If passed, the difference should be zero.
    - The second test uses the two different dates as the parameters to test for the function when there is a defined number of day differences between the dates. If passed, the difference calculated should be equivalent to the expected output, to which in this case 4550 days.
    - Otherwise, if either one or both of the tests fail, an error message “number_of_days_for_all_months test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function
- getRewardsList()
    - A function to return the list of rewards in the form of a vector<string>. If the returned vector is not identical to the assigned strings, such as “5 points - Have a little treat, like an ice cream or a block of chocolate!”, the test would fail. Otherwise the function would return the exact same copy of strings in the vector assigned in the program

HighPriority class :

Under the filename <HighPriorityTester>
A total of 5 tests to demonstrate the usability of each behaviour in the program

- setRewards()
    - Five test cases with different number of days difference are used to test on the program.
    - A defined variable of the class type “RewardsMaster” is present to call the function setDate() to allocate separate dates to calculate the difference in days between two dates

- A defined variable of the class type “HighPriority” is present as well to calculate the difference in days using the inherited difference_in_time function, and also adds the total number of coins the user earned
- The first test uses the differences of two equal dates as the parameters to test for the function when there are no day differences between the dates. If passed, the difference should be zero and the number of coins should be 10.
- The second test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (5 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 20 coins.
- The third test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (10 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 30 coins.
- The fourth test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (1415 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 30 coins.
- The fifth test uses the differences of two different dates as the parameters to test for the function when there is a invalid number of day differences (-1 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 10 coins.
- Otherwise, if either one or all of the tests fail, an error message “HighPriority.cpp test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function

MediumPriority class :

Under the filename <MediumPriorityTester>
A total of 5 tests to demonstrate the usability of each behaviour in the program

- setRewards()
    - Five test cases with different number of days difference are used to test on the program.
    - A defined variable of the class type “RewardsMaster” is present to call the function setDate() to allocate separate dates to calculate the difference in days between two dates
    - A defined variable of the class type “MediumPriority” is present as well to calculate the difference in days using the inherited difference_in_time function, and also adds the total number of coins the user earned
    - The first test uses the differences of two equal dates as the parameters to test for the function when there are no day differences between the dates. If passed, the difference should be zero and the number of coins should be 5.
    - The second test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (5 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 10 coins.

- The third test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (10 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 20 coins.
- The fourth test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (1415 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 20 coins.
- The fifth test uses the differences of two different dates as the parameters to test for the function when there is a invalid number of day differences (-1 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 5 coins.
- Otherwise, if either one or all of the tests fail, an error message “MediumPriority.cpp test failed!” would be outputted to warn the user of a failed test and inform them to review the particular function

LowPriority class :

Under the filename <LowPriorityTester>
A total of 5 tests to demonstrate the usability of each behaviour in the program

- setRewards()
    - Five test cases with different number of days difference are used to test on the program.
    - A defined variable of the class type “RewardsMaster” is present to call the function setDate() to allocate separate dates to calculate the difference in days between two dates
    - A defined variable of the class type “LowPriority” is present as well to calculate the difference in days using the inherited difference_in_time function, and also adds the total number of coins the user earned
    - The first test uses the differences of two equal dates as the parameters to test for the function when there are no day differences between the dates. If passed, the difference should be zero and the number of coins should be 0.
    - The second test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (5 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 5 coins.
    - The third test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (10 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 10 coins.
    - The fourth test uses the differences of two different dates as the parameters to test for the function when there is a defined number of day differences (1415 days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 10 coins.
    - The fifth test uses the differences of two different dates as the parameters to test for the function when there is a invalid number of day differences (-1

days) between the dates. If passed, the coins calculated should be equivalent to the expected output, to which in this case 0 coins.
    - Otherwise, if either one or all of the tests fail, an error message "LowPriority.cpp test failed!" would be outputted to warn the user of a failed test and inform them to review the particular function

User class :

Under the filename <UserTester>
A total of 5 unit tests and 4 automated tests used to demonstrate the usability of each behaviour in the program

- setName() and getName()
    - Two test cases with names are used to test on the program.
    - A default value of "" is set in the constructor to ensure that the name is always an empty string when no input is entered.
    - A string is given in the first test, while an empty string is used for the test. If passed, the name object from the User class should be identical to the entered string in the first test whereas the name would be stated as "None" in the second test
    - Otherwise, if either one or both of the tests fail, an error message "LowPriority.cpp test failed!" would be outputted to warn the user of a failed test and inform them to review the particular function
- addCoins, removeCoins and getCoins functions
    - Some unit tests and automated testing in input05.txt shows the ability of the functions to add coins based on the difference in completion and due date, to deduct coins based on the selected rewards and to get the total amount of coins left for the particular user
- addTask(), deleteTask(), moveTask(), toDolist(), inProgresslist() and completeList()
    - Four automated testing are conducted using the text files to depict the task allocations when tasks are added, deleted and moved to a different list
    - The first test, named "input01.txt" demonstrates the functions' behaviours of adding a task entered by the user by using the "push_back" function in vector. The expected output is stored in the "output01.txt"
    - The second test, named "input02.txt" demonstrates the functions' behaviours of adding and then deleting a task entered by the user by using the "erase" function in vector. The expected output is stored in the "output02.txt"
    - The third test, named "input03.txt" demonstrates the functions' behaviours of moving a task entered by the user to the assigned list. The expected output is stored in the "output03.txt"
    - The fourth test, named "input04.txt" demonstrates the functions' behaviours of adding, deleting and moving multiple tasks entered by the user individually without errors. The expected output is stored in the "output04.txt"

Schedule plan

We plan to complete the coding next week along with test files. We also planned to tie up loose ends and review the entire program by this week after the practical session to ensure

everything has been done as well as expected. The team may meet up online through Discord and platforms such as Google Drive to discuss any potential issues and/or bugs that faced and may potentially faced in the program. Hence, in this week we hope to complete our program after the practical session and if time permits, we may add some interesting features, such as the ability to save the list in a separate text file into the program.

The allocation of tasks among the team are stated below:

Jolene:

- In charge of the main function and the overall design of the to-do list program
- Aims to debug and neaten the code to ensure that all classes interact as expected and are proven functional when applied to the main function before handling the program to Suzanne for testing purposes.
- Established the skeleton code of High, Medium and Low Priority classes, as well as laying the foundation of the Makefile and ensures its functionality throughout the break
- May include additional functions that consist of logic behaviours (such as if and switch statements that would be used in the program) if it helps to improve the clarity of the main program and the overall output.
- Help to integrate all classes to ensure that all objects and behaviours are well-allocated despite facing different coding styles among the team.
- Implement the idea and possible layout of establishing the calculation of coins in the RewardsMaster as a team, which helps to increase the project's complexity
- May include the functionality of saving the program into a text file into the program to enhance user's experience.

Suzanne

- In charge of all the testings and RewardsMaster class, as well as integrating the Priority classes into the main program and other classes programmed by Jolene.
- Aims to construct all unit test cases, automated and regression testing once the main program is completed. This would ensure that all classes interact as expected and are proven functional when applied to the main function
- Established the skeleton code of User and Task classes, as well as constructing the foundations of the RewardsMaster class based on the reference of numerous calendar program online throughout the break
- May include and suggest additional functions that consist of logic behaviours (such as if and switch statements that would be used in the program) if it helps to improve the clarity of the classes and the overall output.
- Ensures that the RewardsMaster can be inherited from the three priority classes, as well as being implemented flawlessly in the main program
- Implement the idea and possible layout of establishing the calculation of coins in the RewardsMaster as a team, which helps to increase the project's complexity
- Help in providing feedback and research on solutions on existing and potential bugs in the program.

- Plan to construct the integration testing in the form of a text file to test the output of the functions as well as the main program with the expected output.

Expected visual of user screen:

Enter your name: (request input)

Main menu:
What would you like to do today?
1 - View To-Do list
2 - View In-Progress list
3 - View Complete list
4 - Create a task
: (request input)

(After selection - If user chooses to view list)
Here is your ___ list:
List of tasks

Select a task:
: (request input)

(After selection - if user chooses to create a task)
How important is the task? Choose priority:
1 - High
2 - Medium
3 - Low
: (request input)

(After selection - if user selects a task)
Choose from the following options:
E - edit task
D - delete task
M - move task (eg from to-do to complete)
: (request input)

Edit task options
1 - edit title
2 - edit description
3 - edit due date
4 - edit checklist (array of mini tasks related to main task)
5 - edit notes
6 - edit priority