Interpreted requirements of the program

1. Create Employee class
   1. **Data members**: Employee ID, name, date of birth, address, phone number, email address, department number and basic pay
   2. **Basic functions**: setFields (), calculatePay (), calculateTax (), displayData (), getID (), and getBasicPay ()
2. Create Ordinary Employee class (*inherits members of Employee class)*
   1. **Additional data members:** hourly overtime pay rate, no of overtime hours, and yearly bonus
   2. **Additional functions:** setFields () *new,* calculatePay () and displayData () *overrides the Employee class*
   3. **Overloaded constructors:** OrdinaryEmployee object can be instantiated with either (i) yearly bonus or (ii) hourly overtime pay rate, overtime hours and yearly bonus
3. Create Senior Employee class (*inherits members of Employee class)*
   1. **Additional data members:** performance bonus
   2. **Additional functions:** setFields () *new,* calculatePay () and displayData () *overrides the Employee class*
4. Prints a menu that provides the following functionalities:
   1. Input Employee’s profile (including Employee ID, name, date of birth, address, phone number, email address, department number, basic pay, gross pay)
   2. calculate its employee’s gross pay (taking into consideration overtime pay and bonus)
   3. calculate its employee’s CPF contribution as well as the employer CPF’s contribution
   4. calculate its employee’s Tax contribution
   5. display each employee’s information
   6. display a summary of the total amount of money made towards the payment of salary for all the employees
5. The program shall catch multiple exceptions by implementing try and catch blocks
6. Add on new features

Diagram / Illustrations of program design

I have used maps to store and to index by some of their attributes.

Pointer to Employee Object

Employee ID

An element of map sorted by Employee ID

Employee name

Pointer to Employee Object

An element of a multimap sorted by Employee name

Pointer to Employee Object

Employee gross pay

An element of a multimap sorted by Employee gross pay

Summary of implementation of each module in my program

* Enter employee’s profile
  + Ask if the user wants to input a senior or ordinary employee’s profile
  + Prompts the user for input for the respective data
  + Prompts user to re-enter if input is not valid
  + Allocates a memory location to store the newly created Employee object
  + Calculates the pay and tax
  + Updates the maps with a pointer to that memory location
  + Updates the employee count and total gross salary
* Display all employees
  + Ask the user which attribute he/she wants the records to be sorted by and the order of which it is sorted
  + Displays all the Employee objects in the choosen order
  + Displays total number of employees registered
* Find an employee's profile
  + If there are currently no employees registered, system informs the user
  + Or else prompts the user to how he/she wants to search (by ID, name or gross pay)
    - If user searches by ID, system prompts the user to enter the ID and outputs the employee profile that matches the ID, or tells the user if there is no match found
    - If user searches by name, system prompts the user for the name of the employee and outputs employee profile(s) that matches the name, or tells the user if there is no match found
    - If user searches by gross pay, system prompts the user for the lower and upper bound gross pay and displays all the employees that have gross pays in between the range
* Display summary of payment of salary for all the employees
  + Displays the list of gross salaries of all senior employees and the total salary paid for all senior employees
  + Displays the list of gross salaries of all ordinary employees and the total salary paid for all ordinary employees
  + Displays total salary paid for all employees
* Change employee's profile
  + Informs user if there are no employees currently registered
  + If there are, prompts the user for the ID of the employee he/she wants to change
  + Asks the user which attribute to change and the new value of the attribute
  + Saves the record and updates the maps if there is a need to
* Delete employee’s profile
  + Informs the user if there are no employees currently registered
  + Or else prompts the user to enter the ID of the employee he/she wishes to delete
  + Displays the employee profile of the employee and prompts the user to confirm his/her deletion of the object, or informs the user if no matches found
  + If user confirms deletion, deletes the employee object and updates the maps and multimaps
  + Informs the user if records are empty after deletion
* View statistical data
  + Displays total ordinary employees, total senior employees and total employees altogether
  + If the number if senior employees are not zero, displays the maximum, minimum and average senior employees gross pay
  + If the number if ordinary employees are not zero, displays the maximum, minimum and average ordinary employees gross pay
  + If total number of employees is not zero, displays the average employee gross pay.

Reflections of program development

**Difficulties faced**

I was initially looking for a type of C++ container object that would can be sorted by different attributes of an object. Vectors, deques and list would only allow to be sorted by one attribute, sets would be more efficient in sorting but also only allow sorting in one way, that is to overload the < and == operators. Then I found maps which allows sorting by the first element of a pair while the second element contains the data. This allows for much efficient sorting and retrieval of an object with many keys. I implemented 3 maps (2 are actually multimaps as they contain non unique keys) as shown in page 2, this way it would be much easier to display and search for data according to the keys provided in the maps.

I was considering storing senior and ordinary employee objects into different containers at first but however decided to combine it to one container for simplicity. The use of pointers of the parent class has greatly aided the efficiency of the program. Since pointers of the base class can be used on derived objects, it enables us to combine senior and ordinary employee objects into one container by storing their homogeneous base class pointer. Also it greatly saves memory space. (Imagine implementing the maps on page 2 that stores an employee object instead of a pointer to it, it would make the maps much more “bulky”!)

**What could have been done better / Possible enhancements in future**

Due to time constraint there are many aspects that could be further improved. We could create more indexes (maps) on more of the attributes. This would give the user more ways to search for an employee and also allows it to be displayed in more ways. We could create an advance search which takes in more than one attributes from the user and search for the employee object. We could export the information into a text file or binary file to store or print out the information when we need to, or we could also input information from there rather than keying in manually. We could allow conversion from an ordinary to a senior employee or vice versa. We could also create new attributes like date created, date last modified to document our records.

**What I have learnt**

In the course of the assignment, I have learnt more about maps, multimaps and their functions, iterators, reverse iterators. I have also learnt about inheritance, base class, derived class, virtual functions and function overriding the in derived class. I have learnt more about exception handling through try and catch blocks and how useful they can be, especially when they can bypass intermediate functions to go to the main function that calls it (or has the catch block).