Q1 . Construct three classes LinkedList (*integers as its elements*), SortList and LinkedListOpertion with the following operations as shown below *[No inbuilt function allowed for Linked list and Sort function]*).

- LinkedList consists of two operations
  addList() #Add elements into linked list
  dispList() #Display The linked list

- SortList inherits from LinkedList and consists of three operations
  sortList() #Returns the sorted version (ascending order) of the original linked list.
  dispList() #Display the sorted version (ascending order) of the original linked list
  addList() # Returns a list which is the result of position wise addition of the elements in the original unsorted list and the sorted list.

- LinkedListOperation inherits from SortList and consists of two operations
  removeDuplicates() #Returns a list after removing duplicates from the sorted version of the original list
  middleElement() #Display the middle element in the resultant list, after removing duplicates from the sorted version of the list. If the number of elements in the resultant list is odd, display the middle element; if it is even then display the average (rounded to two decimal places) of the two middle elements.

Create a main class DemonstrateLinkedList to perform the following operations.

1. Create a Linked list of N elements (We can increase the size of N)
2. Add elements to the created list
3. Sort the Linked List in ascending order
4. The sum of pairwise addition of the elements in the original unsorted list and the sorted list.
5. Remove duplicates from the list
6. Find the middle element from the list after removing duplicates.

Input format

The first line of the input is the size of the LinkedList
On the following line you can use

- *A* to add/insert a new element into the end of the Linked List (format:- A<value>)
- *I* to increase the size of the Linked List (format:- I <value>)
- *d* to print the original unsorted list (format:- d)
- *S* to sort the linked list, and then s to display the sorted list (format:- S s)
- *R* to find the sum of the unsorted and sorted list, and then r to display the resultant list (format:- R r)
- *P* to remove the duplicate, and then p to display the resultant list (format:- P p)
- M to find the middle element from the list after removing duplicates, and m to print the middle element value (format:- M m)
- E to exit the program

Output format

- On the first line, if the value specified is not a positive integer > 0, print “InvalidSIZE” and terminate the program.
- If the list is empty and in the input line if the options d or S s or R r or P p or M m is used, then print “ListEmpty”

- If the list is full and in the input line if the option A is used, then print “ListFull”
- In the input line if the option *I* is used, then after increasing the size of the Linked List print “Increased”
- If the input line is d, then print the values in the linked list separated by space
- If the input line is S s, then print the values in the sorted version of the linked list separated by space
- If the input line is R r, then print the values in the resultant list linked list separated by space
- If the input line is P p, then print the values in the resultant list linked list separated by space
- If the input line is M m, then print the resultant value (middle element after removing duplicates)
- **If the input line is E, then print “End”, and terminate the program**

Constraints (Taken for granted: no need to check inside program)

The elements in the list are positive integers

Sample Input and Output

INPUT	***OUTPUT***
3 *A 21* *A 5* *A 30* *d* *S s* *A 10* *I 5* *A 21* *d* *S s* *R r* *P p* *d* *M m* *A 24* *P p* *M m* *E*	*21 5 30* *5 21 30* *ListFull* *Increased* *21 5 30 21* *5 21 21 30* *26 26 51 51* *5 21 30* *21 5 30 21* *21* *21 5 30 21 24* *5 21 24 30* *22.50* *End*

Q2 We are building a cricket app that notifies viewers about the information such as current score, run rate and predicted score on a cricket match having 50 overs (1 over have 6 balls). In order to build the app, we are going to make use of three classes

- **CricketOverDetails :-** (Attributes :- *overNumber*, *runsScoredInTheOver*, *wicketsFallenInTheOver*).
- **CricketDataManager** :- Stores the details of each over entered, and have the functionalities
    - getCurrentScore :- returns the current score in the format (display format as follows) TotalRuns/LatestOver(TotalWicketsFallen)
    - updateScore :- Add/Insert details of a particular over (takes parameters :- *overNumber*, *runsScoredInTheOver*, *wicketsFallenInTheOver*)
    - getCurrentRunRate :- return the latest run rate (Formulae : RunRate = TotalRuns/LatestOver) rounded to two decimal places.
    - getPredictedScore :- returns the predicted score (Formulae : PredictedScore = RunRate*TotalNumberOfOvers). If the calculated predicted score is a floating point value, then return the floor value.

  TotalRuns is the sum of *runsScoredInTheOver* in all successfully added cricket overs
  LatestOver is the highest *overNumber* in all successfully added cricket overs
  TotalWicketsFallen is the sum of *wicketsFallenInTheOver* in all successfully added cricket overs
  TotalNumberOfOvers is the maximum allowed overs in the match

- **AppTester** :- It is the main class, and it interacts with CricketDataManager class to
    - Feed the details of a particular over
        - When you add details of an over, you will not be able to add details of a particular over which has the *overNumber* less than or equal to that of the last entered *overNumber*. Besides that *overNumber* should be less than 51
        - When you add details of an over, you will not be able to add details of a particular over which has the *runsScoredInTheOver* > 36
        - When you add details, the total number of wickets fallen (including *wicketsFallenInTheOver* in the current entry detail) should not be greater than 10
    - Displaying the current score
    - Displaying the latest run rate
    - Displaying the predicted score

Constraints (Taken for granted: no need to check inside program)

- overNumber, an integer > 0
- runsScoredInTheOver, an integer >= 0
- wicketsFallenInTheOver, an integer >= 0

Input format

Input is given as a space separated list in the format: <Character> <Zero or more Parameters>
<Character> can be (A or C or R or P)

- *A : If the <Character> is A then,* <Zero or more Parameters> will be having three integers separated by space representing *overNumber*, *runsScoredInTheOver* and *wicketsFallenInTheOver*
- *C* : *If the <Character> is C then,* <Zero or more Parameters> will be empty. This option is used to display the current score
- *R*: *If the <Character> is R then,* <Zero or more Parameters> will be empty. This option is used to display the latest run rate.
- *P*: *If the <Character> is P then,* <Zero or more Parameters> will be empty. This option is used to display the predicted score

Output format

- *If the <Character> in input is A then*
  - (*overNumber* <= to the last added *overNumber*) or (*overNumber* > 50), then display "InvalidOVER"
  - *runsScoredInTheOver* > 36, then display "InvalidRUNS"
  - Total number of wickets fallen including the current *wicketsFallenInTheOver* > 10, then display "InvalidWICKETS"
  - In all other cases, the cricket over details will get added, and then display "Added"

  [The checks should be done in the order mentioned, and for a given detail it will display only one message among { InvalidOVER, InvalidRUNS, InvalidWICKETS, Added}, which ever satisfies first]

- *If the <Character> in input is C then,* display the current score in the format TotalRuns/LatestOver(TotalWicketsFallen).
  - If no over details are added yet then display "MatchNotStarted"
- *If the <Character> in input is R then,* then display the latest run rate rounded to two decimal places.
  - If no over details are added yet then display "MatchNotStarted"
- *If the <Character> in input is P then,* display predicted score. If the calculated predicted score is a floating point value, then display floor value.
  - If no over details are added yet then display "MatchNotStarted"

Input	**Output**
A 2 40 0 C A 1 4 0 A 3 5 1 A 4 4 2 A 2 7 0 A 6 7 1 A 8 12 1 P R C A 10 5 1 A 12 4 2 A 13 5 3 A 15 4 1 R	InvalidRUNS MatchNotStarted Added Added Added InvalidOVER Added Added 200 4.00 32/8(5) Added Added InvalidWICKETS Added 3.00

Q3. There are a number of different kinds of accounts that a bank supports.

- **SavingsAccount -** There is 7% interest paid monthly. There is no minimum balance required. ***Example:)*** *If Balance =50 , added interest balance on the month end is 53.5*
- **CheckingAccount -**
  - ➔ There is 7% interest paid monthly.
  - ➔ This account charges a fee at the end of the month: 10% of the added interest balance.
  - ➔ There is a penalty of 10.00 at the end of the month if the balance (after deducting fee on the added interest balance) falls below a minimum of 100.00.

    Example:) *Balance =50 , added interest balance is 53.5, after fee deduction balance is 48.15, and final balance after penalty is 38.15.*

Create a class **BankAccount**. Create classes **SavingsAccount** and **CheckingAccount** as the subclasses of BankAccount**.** There is a method getMonthEndBalance in each of these classes (Identify the type of polymorphism and use in the program).

- In BankAccount class, getMonthEndBalance just returns the balance amount
- In SavingsAccount class, getMonthEndBalance returns the added interest balance amount
- In CheckingAccount class, getMonthEndBalance returns the balance amount after applying the interest rate, then applying account charge and then applying the penalty

Constraints

Balance is a non negative number.

Input Format

The input contains the accountType and balance separated by colon.

balance should be>= 20.
accountType is either *SavingsAccount* or *CheckingAccount*

Output Format

The output prints the result of getMonthEndBalance based on the accountType (Final result must be rounded to two decimal places.)

If balance <= 20, then print "*overdraft*".
If accountType is not *SavingsAccount* or *CheckingAccount,* then print "invalidAccount"

SAMPLE INPUTS	**SAMPLE OUTPUTS**
CheckingAccount:50	*38.15*
SavingsAccount:100	*107*