*CSE 102*

**Classes as Composite Data Types**

1. Say that you want to know minimum, maximum, mean, median, mod, and variance of a numeric data provided to you as an integer array. Observe that if you write a separate function for each of these tasks, you will end up traversing the array again and again (at least 6 times here). Even though it is good to have a function for a single conceptual task, sometimes it can lead to severe performance penalties. If in your program whenever you need one of these stats you also need the others then it’s reasonable to go for the following way of doing things.

Write a single function which takes the integer array and computes all these metrics and return them as an instance of a class.

**class** Stats {

**int**[] data;

**int** min, max, mod;

**double** median, mean, variance;

}

**static** Stats analyze(**int**[] data) {

// your code

}

1. Write a function which takes an array of students, calculates each student’s letter grade and updates the *letterGrade* field according to the following scheme.

weightedAvg = 20% homeworkGrade + 30% midtermGrade + 50% finalGrade

letterGrade = A if weightedAvg >= 90

B if 90 > weightedAvg >= 70

C if 70 > weightedAvg >= 50

F if 50 > weightedAvg

**class** Student {

String name;

**double** homeworkGrade;

**double** midtermGrade;

**double** finalGrade;

**char** letterGrade;

}

Say that you want to imitate LinkedList<String> structure using your own constructs. You defined the Node class to represent a single item in your list.

**class** Node {

String data;

Node next;

}

1. Write a function which takes a Node representing the head (first element) of a list and an integer *n* as parameters, and returns *nth* element of the list (starting from 0). Your function should return null when *n* is too big or a negative value. You can assume that there exists an element in this list whose *next* field is null.
2. Write a function which takes a Node representing the head (first element) of a list and a string *s* as parameters and removes the first occurrence of *s* in the list if it’s present. Your function should return true iff at least one *s* was in the list and deleted successfully. You can assume that there exists an element in this list whose *next* field is null.
3. Write a function which takes a Node representing the head (first element) of a list, a string *s* and an integer *n* as parameters, and inserts a new Node which refers to the string *s* to the list in a way that after the insertion is done the new element is at *nth* position (starting from 0). Your function should return true if the operation is successful and false if the value of *n* is invalid. You can assume that there exists an element in this list whose *next* field is null.
4. Write a function which takes a Node representing the head (first element) of a list, and returns true iff the list has a linear structure, that is, it does not have any loops in it.

**e.g.**

Contains a loop, your function should detect it and return false.