2023417682063

Semester End Examination
MCAC 101: Object Oriented Programming
Unique Paper Code: 223421111
Semester I
December 2024
Year of Admission: 2024

Time: 3 Hours **Max Marks:70**

Notes: Document your code. Follow a modular approach, where applicable. Make suitable assumptions, if required.

1. A text file contains information about students in a class. The student information includes roll number, name, and marks in five courses having the course codes: 101, 102, 103, 104, 105. Course name is not required. Write C++ code to perform the following:

   a. Create a class `Student`, comprising the following information: roll number, name, and grades in five courses (to be computed as per the criteria given below).
   b. Without storing the data about all students in an aggregate structure, read a text file comprising information about the students, and compute average marks (say, mu) for each course. For each student, convert marks to grade in each course as per the following criteria:
      i. (marks obtained >= 90) or marks obtained >= 1.40*mu): O grade
      ii. (80 <= marks obtained < 90) or (1.25*mu <= marks obtained < 1.4*mú): A grade
      iii. (70 <= marks obtained < 80) or (1.10*mu <= marks obtained < 1.25*mu): B grade
      iv. (60 <= marks obtained < 70) or (0.90*mu <= marks obtained < 1.10*mu): C grade
      v. (50 <= marks obtained < 60) or (0.70*mu <= marks obtained < 0.90*mu): D grade
      vi. All other cases: F grade
   c. Produce a file of `Student` objects.

(15)

2. Using the **map** feature of C++, write a function which accepts the name of a month as input and returns its position in the calendar. For example, if the input to the function is **"June"**, it should return **6.**

(5)

3. Write a recursive function that accepts as input a vector of floating point numbers and returns the number of occurrences of each floating-point number in the vector. Use a suitable data structure. If a number appears multiple times, include it only once in the function output.

(10)

4. The similarity between two floating point vectors **x** and **y** is often computed as $\frac{x.y}{\|x\|.\|y\|}$.
   To compute **x.y**, you just need to find the sum of element wise products. For example, the product of vectors **[1, 2, 3]** and **[4, 5, 6]** is computed as the sum **1*4 + 2*5 + 3*6**. Further, the norm $\|x\|$ of a vector **x** is computed as the square root of **x.x**. Write a recursive C++ function that computes the dot product of two vectors. Use it in another function to find the similarity between two vectors. **(10)**

5. Show the runtime stack for the first 6 calls of **hanoi()**, when it is invoked as **hanoi(3, 1, 2, 3)**. Also, rewrite the function **hanoi()** without using recursion. **(15)**

```
#include <iostream>
#include <stack>
#include <cassert>
using namespace std;

// Objective: Use recursion to solve the Tower of Hanoi problem
// Input parameters:
//    - nDisks: Number of disks to be moved.
//    - source: Source pole identifier.
//    - spare: Spare pole identifier.
//    - destn: Destination pole identifier.
// Side effects: Outputs each move to console
// Return value: None

void hanoi(int nDisks, int source, int spare, int destn) {
    assert(nDisks >= 1); //validate number of disks
    if (nDisks == 1) {
        cout << "Move a disk from pole " << source <<
                " to pole " << destn << endl;

    }
    else {
        // Move n-1 disks from source to spare
        hanoi(nDisks - 1, source, destn, spare);
        // Move the nth disk from source to destination
        cout << "Move a disk from pole " << source <<
                " to pole " << destn << endl;
        // Move the n-1 disks from spare to destination
        hanoi(nDisks - 1, spare, source, destn);
    }
}

int main() {
    int nDisks;
    cout << "Enter the number of disks: ";
```

```
    cin >> nDisks;

    // Move nDisks from pole 1 to pole 3 using pole 2 as a spare
    hanoi(nDisks, 1, 2, 3);

    return 0;
}
```

6. Write a recursive function which accepts two vectors of integers (each of which is arranged in ascending order) as input and returns the merged vector, leaving the original inputs unchanged. Show the runtime stack for the vectors `[10, 20]` and `[12, 15, 30, 40]`. **(15)**