Birla Institute of Technology and Science, Pilani Hyd Campus
BITS F232: Foundations of Data Structures and Algorithms
1st Semester 2023-24 Lab No:3

General Tips

- Try to use functions as much as possible in your code. Functions increase reusability and the pass-by-value feature provides a significant help sometimes. Modularizing your code also helps you to debug efficiently.
- The use of STL is strictly prohibited in this lab. You should not use any inbuilt data-structure/ algorithm to do any task.
- Indent your code appropriately and use proper variable names, if not already provided in the question. Also, use comments wherever necessary.
- Make sure to debug your code after every task or after every code block to avoid having to debug your entire file at once in the last minute.

Overview:

We have already covered basics of C++ programming and are well versed will Object Oriented Programming concepts like Encapsulation, Abstraction, Inheritance, Polymorphism etc. in the previous two labs. For this lab, our goal is to learn about Arrays which are concrete data structures. You must have learnt Arrays in computer programming course. However, in this lab we will learn how to use Arrays to build an application using Objects that will automatically adjust its' size. While C++ STL has a vector as a sequence container (data structure that can store similar types) that can change its' size dynamically, in this lab you will implement your own dynamic array without using vectors.

Program 1 (DynamicArray.cpp attached):

This task is about implementing a dynamic array class with member functions for insert, delete, search, sort, grow, shrink etc. The below C++ code fragment has complete functions and classes for all the required objects as we discussed in the class. The complete cpp file is given in the class page (DynamicArray.cpp). You should see the output as given below:

Output:

```
5 3 11
5 7 3 11
3 5 7 11
3 5 7 11 15 16
6
3 5 7 15 16
3 5 7 15
5 7 15
5 15
2
```

Program 2 (GameEntry.cpp as discussed in the class is attached):

This task is about the GameEntry class discussed in the class. Pl. refer to the GameEntry.cpp source file uploaded along with this lab sheet. Output of the run is as shown below (as discussed):

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Rohit 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Virat 95
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 120
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
3
Gill : 120
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 200
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
3
Gill : 200
Gill : 120
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

Task 1:

Modify the code given so that you will have another option for printing **how many number of entries are there for each player** at any point in time during the run. The code fragment to complete this task is given partly as below with comments. Your task here is to understand the code fragment and complete the missing statements. If you want to use a different logic to implement the task, you are welcome to do so.

```
void Scores::printPlayersCount() {
    // isPicked is a table that helps us record whether a player was already
       considered for calculating his/her count.
    bool isPicked[numEntries]{false};

    for (int i = 0; i < numEntries; i++)
    {
        if (isPicked[i]) // this player was considered previously
            continue;

        string playerName = entries[i].getName();
        int playerCount = 1;
```

```
        for (int j = i + 1; j < numEntries; j++)
        {
            if (isPicked[j]) // this player was considered previously
                continue;

            //Compare the name of ith player with jth player and if they are
              equal then increment the player count for ith player and also
              update the isPicked table's flag for jth player to True. If
              they are not equal, then skip and go to next jth player (Task1)
        }
        cout << playerName << " : " << playerCount << endl;
    }
}

// checks whether two strings are equal or not
bool Scores ::isEqual(string a, string b)
{
    if (a.length() != b.length())
        return false;
    // check if a[i] is equal to b[i] through a loop and return false if not
(Task1)
    return true;
}

void Scores::printAllScores()
{
    for (int i = 0; i < numEntries; i++)
    {
        cout << entries[i].getName() << " : " << entries[i].getScore() <<
"\n";
    }
}
void showOptions()
{
    cout
        << "1:  Add Player\n"
        << "2:  Remove Player By Index\n"
        << "3:  Print Scores\n"
        << "4:  Print Players Count\n"
        << "5:  Exit\n";
}
int main()
{
    Scores scoresObj;
    int option;
    string playerName;
    int score;

    while (1)
    {
        showOptions();
        cin >> option;
        switch (option)
        {
```

```
        case 1:
            cout << "Enter Player Name and Score\n";
            cin >> playerName >> score;
            scoresObj.add(GameEntry(playerName, score));
            break;
        case 2:
            int index;
            cout << "Enter the index\n";
            cin >> index;
            scoresObj.remove(index);
            break;
        case 3:
            scoresObj.printAllScores();
            break;
        case 4:
            scoresObj.printPlayersCount();
            break;
        case 5:
            return EXIT_SUCCESS;
        }
    }
}
```

You should get the output as shown below:

```
4
Gill : 2
Virat : 1
Rohit : 1
1:  Add Player
2:  Remove Player By Index
3:  Print Scores
4:  Print Players Count
5:  Exit
```

Task 2:

Modify the above code to display unique entries for each player. Part solution is given in GameEntry_Unique.cpp. You should get the output as shown below:

```
3
Gill : 200
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

Task 3:

Modify the code given so that you will have another option for printing **the name of players having sum of all entries in a specified range** i.e. min value <= sum of player's entries <= max value at any point in time during the run. The range of scores (max value, min value) will be taken as input by the user. The code fragment to complete this task is given partly as below with comments. Your task here is to understand the code fragment and complete the missing statements.

```
void Scores::printPlayersInScoreRange(int maxValue, int minValue) {
    // isPicked is a table that helps us record whether a player was already
    // considered for calculating his/her count.
    bool isPicked[numEntries]{false};

    for (int i = 0; i < numEntries; i++)
    {
        if (isPicked[i]) // this player was considered previously
            continue;

        //Initialize the sum with ith player's score(Task 4)
        int sum;

        string playerName = entries[i].getName();

        for (int j = i + 1; j < numEntries; j++)
        {
            if (isPicked[j]) // this player was considered previously
                continue;

            //Compare the name of ith player with jth player and if they are
            //equal then update the sum and also
            //update the isPicked table's flag for jth player to True. If
           //they are not equal, then skip and go to next jth player (Task4)
        }

         //Check if sum of scores of current player is between minValue and
        //maxValue. If yes, then print the player's name and sum (Task 4)
        cout << playerName << " : " << sum << endl;
    }
}

void showOptions()
{
    cout
        << "1:  Add Player\n"
        << "2:  Remove Player By Index\n"
        << "3:  Print Scores\n"
```

```
        << "4:  Print Players Having Sum of Scores in Range\n"
        << "5:  Exit\n";
}

int main()
{
    Scores scoresObj;
    int option;
    string playerName;
    int score;

    while (1)
    {
        showOptions();
        cin >> option;
        switch (option)
        {
        case 1:
            cout << "Enter Player Name and Score\n";
            cin >> playerName >> score;
            scoresObj.add(GameEntry(playerName, score));
            break;
        case 2:
            int index;
            cout << "Enter the index\n";
            cin >> index;
            scoresObj.remove(index);
            break;
        case 3:
            scoresObj.printAllScores();
            break;
        case 4:
            int maxValue, minValue;
            cout << "Enter max value and min value\n";
            cin >> maxValue >> minValue;
            scoresObj.printPlayersInScoreRange (maxValue, minValue);
            break;
        case 5:
            return EXIT_SUCCESS;
        }
    }
}
```

----------*----------**