May 27, 2020

Sabancı University

Spring 2019-2020

CS201 - Introduction to Computing

Final Open-Book Take-Home Exam

Q1	Q2	Q3	Q4	Total
/15	/30	/30	/25	/100

Deadline is 12:00. Please use your time wisely.

- **There will not be any late submissions!**
- **We will not accept any submissions other than SUCourse!**

This link contains details that we shared before in the announcement.

Here is the Google Meet Link for your questions. Please do not wait in the Google Meet room during the whole exam, so that you will not get disturbed by the noise. Whenever you have a question, join the link, ask your question, and then leave.

All related files to be used in the questions (final exam bundle) were shared in this link.

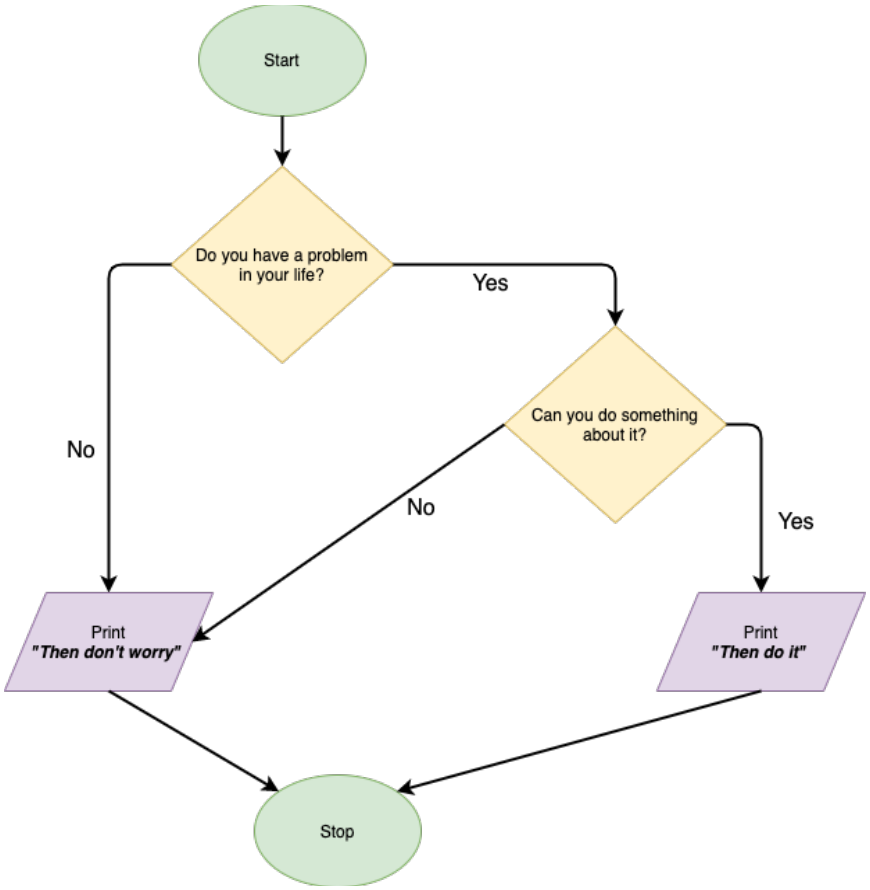
Good Luck!

1. **(15 pts)** Write a C++ program that will implement the following scenario: Your program will ask the user if s/he has any problems in their life. If the user enters "*No*", then your program should display "Then don't worry" and terminate its execution. Otherwise, i.e. if the user enters "*Yes*", then your program should ask the user if s/he can do something about it. If the user enters "*No*", then your program should display "Then don't worry" and terminate its execution. On the other hand, if the user enters "*Yes*", then your program should display "Then do it", and terminate its execution.

   You may assume that all of the inputs will always be entered as either "***Yes***" or "***No***"; thus, you do not need to perform any validity checks.

   Important Note: *Download* "`q1_main.cpp`" file from ***Q1*** folder in the final exam bundle, *modify* this file such that it will include your answer to this question, and then *upload* this file to SUCourse.

   You may find the flowchart of the above described algorithm below.



```
string ans;

cout << "Do you have a problem in your life? ";
cin >> ans;

if (ans == "Yes"){
     cout << "Can you do something about it? ";
     cin >> ans;
     if (ans == "Yes"){
          cout << "Then do it" << endl;
     }
     else{  // else if (ans == "No"){
          cout << "Then don't worry" << endl;
     }
}
else{ // else if (ans == "No"){
     cout << "Then don't worry" << endl;
}
```

OR

```
string ans, res = "Then don't worry";

cout << "Do you have a problem in your life? ";
cin >> ans;

if (ans == "Yes"){
     cout << "Can you do something about it? ";
     cin >> ans;
     if (ans == "Yes")
          res = "Then do it";
}

cout << res << endl;
```

2. **(30 pts)** Write a C++ program that *prints* a ***sub-matrix*** from a given matrix (vector of integer vectors) object. This sub-matrix should be constructed using the ***last*** `rowNum` rows and the ***last*** `colNum` columns of the original given matrix object, where the values of `rowNum` and `colNum` are to be user inputs.

   Please note that you **should** implement the solution of this question ***in parts*** as instructed on the next page.

   Assume that the given matrix is as follows:

1	2	3	4
2	4	6	8
4	8	12	16
8	16	24	32
16	32	48	64

   For instance, if `rowNum` is 2 and `colNum` is 3, then the resulting matrix should be constructed as the last 2 rows and the last 3 columns of the original matrix given above, which is as follows:

16	24	32
32	48	64

   As another example, if `rowNum` is 2 and `colNum` is 2, then the resulting matrix should be constructed as the last 2 rows and the last 2 columns of the original matrix given above, which is as follows:

24	32
48	64

This question continues on the next page → → →

Important Note: *Download* "`q2_main.cpp`" file from ***Q2*** folder in the final exam bundle, *modify* this file such that it will include your answer to this question, and then *upload* this file to SUCourse.

`q2_main.cpp` file includes a function which creates and returns the matrix given above as an example. Therefore, you can directly use this matrix for debugging purposes. However, *this is just a sample matrix*. You should **not** make any assumptions on the number of rows or on the number of columns or on the values in the given matrix; that is, your program should be working correctly in a generic way.

a. **(20 pts)** Write a C++ function that takes a vector of integer vectors (integer matrix) object, the `rowNum` and `colNum` values as its only parameters, and prints a sub-matrix as described on the previous page.

   You may also implement additional functions as well, if required, although it's not mandatory.

```
void printMatrix(const vector<vector<int> > & m){
    for(int i=0; i<m.size(); i++)
        for(int j=0; j<m[i].size(); j++)
            cout << m[i][j] << "\t";
        cout << endl;
    }
}

void returnSubset(const vector<vector<int> > & m, int
rowNum, int colNum){
    vector<vector<int> > result;
    vector<int> temp;
    for(int i=m.size()-rowNum; i<m.size(); i++){
     for(int j=m[i].size()-colNum; j<m[i].size(); j++)
            temp.push_back(m[i][j]);
        result.push_back(temp);
        temp.clear();
    }
    printMatrix(result);
```

```
}
```

OR (next page)

```
void printSubset(const vector<vector<int> > & m, int
rowNum, int colNum){
    for(int i=m.size()-rowNum; i<m.size(); i++){
     for(int j=m[i].size()-colNum; j<m[i].size(); j++)
            cout << m[i][j] << "\t";
     cout << endl;
    }

}
```

b. **(10 pts)** Within the main function of your program, get the `rowNum` and `colNum` values from the user, in this given order, respectively. You may check the sample run on the next page for your convenience.

You may assume that the user will always enter non-negative integer values as inputs. You may also assume that the matrix will not be an empty one. However, you should perform the following checks on the values obtained from the user:

i. `rowNum` should be smaller than or equal to the number of rows in the given matrix, and

ii. `colNum` should be smaller than or equal to the number of columns in the given matrix.

After getting these two values from the user, if at least one of the inputs does not satisfy the above mentioned constraints, then your program should re-prompt for these two values repetitively until both of them are entered correctly.

When these two inputs are obtained successfully from the user, your program should then call the function implemented in part-(a) by using the given matrix and the values entered by the user, so that the respective sub-matrix can be displayed on the console

output.

This question continues on the next page → → →

You may find a sample run below, for the matrix given above:

```
Enter rowNum: 6
Enter colNum: 6
rowNum and/or colNum number is too high!
Enter rowNum: 2
Enter colNum: 5
rowNum and/or colNum number is too high!
Enter rowNum: 6
Enter colNum: 2
rowNum and/or colNum number is too high!
Enter rowNum: 2
Enter colNum: 3
16 24 32
32 48 64
```

```
int rowNum, colNum;
cout << "Enter rowNum: ";
cin >> rowNum;
cout << "Enter colNum: ";
cin >> colNum;

while(rowNum > matrix.size() || colNum > matrix[0].size()){
    cout<<"rowNum and/or colNum number is too high!"<< endl;
    cout << "Enter rowNum: ";
    cin >> rowNum;
    cout << "Enter colNum: ";
    cin >> colNum;
}

returnSubset(matrix,rowNum, colNum);
```

OR (next page)

```
int rowNum, colNum, iter = 0;
do{
   iter++;
   if(iter != 1)
cout<<"rowNum and/or colNum number is too high!"<<endl;

   cout << "Enter rowNum: ";
   cin >> rowNum;

   cout << "Enter colNum: ";
   cin >> colNum;

}while(rowNum>matrix.size()                              ||
                    colNum>matrix[0].size());

returnSubset(matrix,rowNum, colNum);
```

3. **(30 pts)** In this question, you will implement a C++ program that displays the name and birth date information of a number of people, in a predefined order, by reading this information from a text file.

   Please note that you **<u>should</u>** implement the solution of this question ***in parts*** as instructed on the next page.

   You are provided with the following files within the ***Q3*** folder from this final exam bundle: "*date_modified.h*", "*date_modified.cpp*", "*people.txt*", and "*q3_main.cpp*".

   You are already familiar with the Date class which was implemented in "*date_modified.h*" and "*date_modified.cpp*" files. You will find the name and birth date information of different people in the "*people.txt*" file. Each line of this file represents a single person, and contains exactly 4 (four) information, each separated by a single space character (' '). These 4 pieces of information are *name*, *birthDay*, *birthMonth*, *birthYear* values, respectively.

   You can make the following assumptions on the "*people.txt*" file:

   - name will consist only of one single string (even if it contains the last name of the person) like "*MaxVerstappen*".
   - *birthDay*, *birthMonth* and *birthYear* will be provided correctly; meaning that, you don't need to perform any validity checks.
   - Every person occurs only once in the given text file.
   - There will be at least one line in the text file.
   - The filename will always be "*people.txt*".
   - You may assume that the file will be opened correctly; thus, you don't need to check whether it will be opened successfully or not.

   However, you cannot make any assumptions on the number of lines (i.e. number of people) of the given file. Also, note that the content of the given file is not sorted with respect to any given attribute.

This question continues on the next page → → →

Important Note: *Download* all of the "*q3_main.cpp*", "*people.txt*", "*date_modified.h*" and "*date_modified.cpp*" files from ***Q3*** folder in the final exam bundle, *modify* only the "*q3_main.cpp*" file such that it will include your answer to this question, and then *upload* all of these files to SUCourse.

a. **(5 pts)** Create a struct called `Person` in your main file. This struct should contain two values: (i) name of the person (of type string), and (ii) birth date of the person (of type Date class).

```
struct Person{
    string name;
    Date birthdate;
};
```

b. **(10 pts)** Write a C++ function that creates and returns a vector of `Person` struct type that you created in part-(a). This function may or may not take any parameters depending on your algorithm. In this function, you should read all the contents from the input file "*people.txt*", fill the vector you created, and then return that vector.

```
vector<Person> returnVector(){
    vector<Person> people;
    ifstream in;
    string line,name, filename = "people.txt";
    in.open(filename.c_str());
    int day, month, year;
    while(getline(in, line)){
        istringstream iss(line);
        iss >> name >> day >> month >> year;
        Date birth(month, day, year);
        Person p;
        p.name = name;
        p.birthdate = birth;
```

```
        people.push_back(p);
    }
    in.close();        return        people;        }
```

c. **(10 pts)** Write a C++ function that takes a vector of `Person` struct as its only parameter, and prints the contents of this vector in a way that the oldest person is printed the first, while the youngest person is printed the last. In case of a situation such that the birth dates are the same among multiple people, then the alphabetically greater name should be printed first.

   For the given input file, this function should print in the following format:

   JimMatheson 21.3.1960
   DominicMiller 21.3.1960
   AyrtonSenna 21.3.1960
   MichaelSchumacher 3.1.1969
   LewisHamilton 7.1.1985
   MaxVerstappen 30.9.1997
   CharlesLeclerc 16.10.1997

```
void sortAndPrintVector(vector<Person> & people){
  for(int i=0; i<people.size()-1; i++){
    int oldestIdx = i;
    for(int j=i+1; j<people.size(); j++){
     if(people[oldestIdx].birthdate>people[j].birthdate)
        oldestIdx = j;
     else if(people[oldestIdx].birthdate==people[j].birthdate){
       if(people[oldestIdx].name < people[j].name)
         oldestIdx = j;
     }
    }
    Person temp = people[i];
    people[i] = people[oldestIdx];
    people[oldestIdx] = temp;
  }

  for(int i=0; i<people.size(); i++)
```

```
    cout << people[i].name << " " <<
        people[i].birthdate.Day() << "."<<
        people[i].birthdate.Month() << "."<<
        people[i].birthdate.Year() << endl;
}
```

d. **(5 pts)** In the main program, call the functions implemented in part-(b) and part-(c) so that the content of the input file will be printed in the order described above.

```
vector<Person> people = returnVector();
sortAndPrintVector(people);
```

4. **(25 pts)** In this question, you will complete a given C++ program that simulates a sports league by *using* and *modifying* the given `League` class, based on a text file.

   Please note that you **should** implement the solution of this question ***in parts*** as instructed on the next page.

   You are provided with the following files within the ***Q4*** folder in this final exam bundle: "*League.h*", "*League.cpp*", "*q4_main.cpp*" and "*scores.txt*".

   Let's start with the "*scores.txt*" file. Each line of this file represents the score of a different match, and contains exactly 4 information, each of which are separated by a single space character (' '). These 4 pieces of information are *nameOf1stTeam*, *goalsScoredby1stTeam*, *goalsScoredby2ndTeam,* and *nameOf2ndTeam* values respectively.

   You can make the following assumptions on the "*scores.txt*" file:

   - You may assume that the file will be opened correctly; thus, you don't need to check whether it will be opened successfully or not.
   - Both of the team names on each line will consist only of one single string, like "*RealMadrid*" or "*Liverpool*" or "*Schalke04*".
   - A team cannot play with itself. Therefore, there will always be two different team names on the same line.
   - *goalsScoredby1stTeam* and *goalsScoredby2ndTeam* values are provided correctly as non-negative integers, so you don't need to perform any correctness checks.
   - The filename will always be "*scores.txt*" and there will be at least one match (line) in the text file.

   However, you cannot make any assumptions on the number of lines (i.e. the number of matches) of this file. Also, note that the content of the given file is not sorted with respect to any given attribute.

This question continues on the next page → → →

Let's continue with the class files. Here are the details of the header file "*League.h*":

- You will find a `struct` called `team`, which contains 4 different information:
  - `teamName`, of type string, and
  - `points`, `goals_scored`, `goals_conceded`, of type integer.
- In the private part of the `League` class, you will find two data members:
  - `leagueName`, of type string (like "Champions League"), and
  - `allTeams`, of type `team` struct vector, which is described above (*note*: when you create an instance of the class, this vector will be initially empty).
- In the public part of the `League` class, you are given the following member functions:
  - Constructor (**League**): It takes a string value as its only parameter, and creates an instance of that class.
  - **getName**: It returns the `leagueName` of the instance.
  - **sortAndPrintTeams**: It is a void type member function which sorts the `allTeams` vector (private data member) based on the points retrieved by the teams, in descending order, and then prints the point table (standings) on the screen (a.k.a. console output) in a specific format. In case of an equal point situation among the teams, this function displays those teams without any order.

On the other hand, in the "*League.cpp*" file, you may find the corresponding implementations of each member function.

For the purpose of the entire program, you will use the `League` class which was *partly* implemented in "*League.h*" and "*League.cpp*" files.

This question continues on the next page → → →

The main purpose of the *partially given* program is first to create a league, and then to read all the match scores from the "***scores.txt***" file, and finally to print the point table (standings) on the output.

As you probably know,

- A team wins the match if that team scores more goals than its opponent. In such a case, the winning team retrieves 3 (three) points, and the losing team's point remains the same.
- If there is a draw (ie. both teams scored the same number of goals in that match), then both teams retrieve 1 (one) point.

Important Note: *Download* all of the "***q4_main.cpp***", "***scores.txt***", "***League.h***" and "***League.cpp***" files from ***Q4*** folder in the final exam bundle, *modify* all the "***q4_main.cpp***", "***League.h***" and "***League.cpp***" files such that they will include your answer to this question, and then *upload* all of these files to SUCourse.

a. (17 pts) In this part of the question, you will add a void type member function called **`processMatch`** into the `League` class. This function should take exactly 4 parameters: *`nameOf1stTeam`*, *`goalsScoredby1stTeam`*,*`goalsScoredby2ndTeam`*,*`nameOf2ndTeam`*. Within this function, you should decide whether there is a win or draw for the given match information (i.e. based on the values of the parameters), and then update the `allTeams` vector accordingly. You will also update the `points`, `goals_scored`, `goals_conceded` values of the given two teams depending on the score of the match.

Remind that `allTeams` vector is initially empty; meaning that you may need to modify a team that already exists in your vector or you may need to add a new team to your vector with new values.

While implementing the **`processMatch`** member function, you

may implement other member functions as well, although it's not mandatory.

This question continues on the next page → → →

To header

```
void processMatch(string team1, int goal1, string team2, int goal2);
//    int    getTeamIndex(string    tname)    const;    //    optional
```

To cpp

```
int League::getTeamIndex(string tname) const
{
    for(int i = 0; i<allTeams.size(); i++)
        if (allTeams[i].teamName == tname)
            return i; // return its index

    return -1; // there is no such team
}
void League::processMatch(string team1, int goal1, string team2, int
goal2){
    int point1 = 0, point2 = 0;
    if (goal1 > goal2)
        point1 = 3;
    else if (goal1 == goal2)
        point1 = point2 = 1;
    else
        point2                              =                              3;

    int  idx1  =  getTeamIndex(team1),  idx2  =  getTeamIndex(team2);

    if(idx1 != -1){
        allTeams[idx1].points += point1;
        allTeams[idx1].goals_scored += goal1;
        allTeams[idx1].goals_conceded += goal2;
    }
    else{
        team t;
        t.teamName = team1;
        t.goals_scored = goal1;
        t.goals_conceded = goal2;
        t.points = point1;
```

```
        allTeams.push_back(t);
    }
    if(idx2 != -1){
        allTeams[idx2].points += point2;
        allTeams[idx2].goals_scored += goal2;
        allTeams[idx2].goals_conceded += goal1;
    }
    else{
        team t;
        t.teamName = team2;
        t.goals_scored = goal2;
        t.goals_conceded = goal1;
        t.points = point2;
        allTeams.push_back(t);
    }
}
```

b. (8 pts) In this part of the question, you will complete the *partially* implemented main program given in "***q4_main.cpp***".

Here, the main functionality is decomposed into 5 (five) parts, and 4 of them are already implemented in "***q4_main.cpp***":

i. A `League` instance called `league` was created (already implemented).

ii. The file ("***scores.txt***") was opened, and there is no need to check whether it was opened correctly (already implemented).

iii. The file ("***scores.txt***") has to be read and the content should be stored in the `allTeams` private data member of the `League` class via the use of the `processMatch` member function implemented in part-(a) of this question. In other words, you will read all the lines from the "***scores.txt***" file, retrieve the 4 information from each line, and then call the `processMatch` member function. **This step will be implemented/completed by CS201 students.**

iv. The file ("***scores.txt***") was closed (already implemented).

v. The league table (standing) was printed by calling **sortAndPrintTeams** member function (already implemented).

For the given "***scores.txt***" file, when you run the complete code, the output (the point table) should be as follows:

Results for Champions League are as follows:

TeamName	*GS*	*GC*	*PTS*
RealMadrid	*7*	*4*	*7*
Barcelona	*5*	*4*	*6*
Liverpool	*6*	*6*	*4*
BayernMunich	*2*	*6*	*0*

```
string line;
string team1, team2;
int goal1, goal2;
while(getline(in, line)){
    istringstream iss(line);
    iss >> team1 >> goal1 >> goal2 >> team2;
    league.processMatch(team1, goal1, team2, goal2);
}
```