CS201 – Spring 2024-2025
Homework 5 – Family Tree Management System
Due May 23rd, Friday, 22:00
(Late Deadline May 24th 22:00)

Introduction

In this homework, you will build a Family Tree Management System using C++. You will apply your knowledge of classes, file I/O, recursion, and vector operations. The main idea is to read people and their relationships from files, construct the family tree in memory, and perform recursive queries such as finding ancestors and descendants.

Objective

This assignment will help you practice how to:

- Create and use classes with proper encapsulation
- Implement recursive functions for traversing hierarchical data
- Read and process person and relationship data from files
- Navigate family trees to find ancestors and descendants
- Display results through a user-friendly, menu-driven system

Inputs, Flow of the Program, and Outputs

Sample Inputs (name birthDate):

people.txt:

John 1950

Mary 1952

Bob 1975

Alice 1978

Charlie 2000

David 2002

Eva 2005

Frank 2020

…

---

relationships.txt(father mother child):

Adam Eve Abraham

Abraham Sarah Isaac

Isaac Rebecca Jacob

Andrew Emma

…

Child can be null

Class Format

These are examples of headers you can use for the Person and FamilyTree classes;

Person Class

```
#include <string>
#include <vector>
using namespace std;

class Person {
private:
    string name;
    int birthYear;
    string motherName;
    string fatherName;
    vector<string> childrenNames;
```

```
public:
    Person(string n, int by);

    string getName() const;
    int getBirthYear() const;
    string getMotherName() const;
    string getFatherName() const;
    vector<string> getChildrenNames() const;

    void setMotherName(const string& mName);
    void setFatherName(const string& fName);
    void addChildName(const string& childName);
};
```

FamilyTree Class

```
#include "Person.h"
#include <string>
#include <vector>
using namespace std;

class FamilyTree {
private:
    vector<Person> people;
    int findPersonIndex(const string& name) const;

public:
    FamilyTree();

    bool loadFromFile(const string& peopleFile, const string&
relationshipsFile);
    void addPerson(string name, int birthYear);
    bool setParentChild(string parentName, string childName, bool
isMother);

    vector<string> findAncestors(const string& name, int
generations) const;
    vector<string> findDescendants(const string& name, int
generations) const;
```

```
    void displayPerson(const string& name) const;
    void displayAllPeople() const;
};
```

Menu Options

1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit

Flow of the Program:

1. Display Person Information

- Create a function that retrieves and displays a person's complete profile
- The function should accept a person's name as input
- It should output:
    - Name and birth year
    - Parents information (if available)
    - List of children (if any)
- Include appropriate error handling for when a person doesn't exist

2. Find Ancestors

- Implement a recursive depth-first search algorithm to find a person's ancestors
- The function should accept:
    - A person's name
    - Number of generations to search
- It should return a list of ancestors organized by their generation relative to the starting person

- Output should follow the format: "Generation X: [Name]" for each ancestor
- Handle cases where the person has no known ancestors

3. Find Descendants

- Implement a recursive depth-first search algorithm to find a person's descendants
- The function should accept:
    - A person's name
    - Number of generations to search
- Return descendants organized by generation (using depth-first traversal)
- Format output as "Generation X: [Names]" with names in the same generation separated by commas
- Handle cases where the person has no descendants

4. Add New Person

- Implement functionality to add a new person to the family tree
- The function should:
    - Accept a name and birth year
    - Create a new person object
    - Add it to the data structure
    - Handle cases where the person already exists(only one person can exist with a name)

5. Add Parent-Child Relationship

- Implement functionality to establish relationships between existing people
- The function should:
    - Accept parent name, parent type (mother/father), and child name
    - Establish bidirectional relationships in the data structure
    - Handle cases where either person doesn't exist

6. Program Exit

- The program exits with a console output “Goodbye”.

Notes on Depth-First Search Implementation

For the ancestor and descendant search functions, you must implement a depth-first search algorithm:

- Depth-first search explores each branch completely before moving to siblings
- For finding ancestors:
    - Start with the person
    - Recursively explore their mother's lineage completely
    - Then explore their father's lineage
    - Return results organized by generation
- For finding descendants:
    - Start with the person
    - For each child, recursively explore all of their descendants before moving to siblings
    - This will result in output organized by family branches rather than strictly by generation level

Your implementation should handle edge cases gracefully, including non-existent people, empty trees, and requests for zero generations.

Following is the findDescendant function that uses this search algorithm to find the descendants of the given person, you can use this function directly for finding the descendants:

```
vector<string> FamilyTree::findDescendants(const string& name, int
generations) const {
    vector<string> result;

    // Check if person exists
    int personIndex = findPersonIndex(name);
    if (personIndex == -1 || generations <= 0) {
        return result; // Return empty vector - main.cpp will
handle error message
    }

    const Person& person = people[personIndex];
    vector<string> childrenNames = person.getChildrenNames();

    // Add immediate children as Generation 1
    if (!childrenNames.empty()) {
        string genStr = "Generation 1: ";
        for (size_t i = 0; i < childrenNames.size(); i++) {
```

```
            genStr += childrenNames[i];
            if (i < childrenNames.size() - 1) {
                genStr += ", ";
            }
        }
        result.push_back(genStr);

        // Recursively find descendants for each child
        if (generations > 1) {
            for (const string& childName : childrenNames) {
                vector<string> childDescendants =
findDescendants(childName, generations - 1);

                // Increase generation numbers
                for (const string& descendant : childDescendants) {
                    int genNum = stoi(descendant.substr(11,
descendant.find(':') - 11));
                    string newDescendant = "Generation " +
to_string(genNum + 1) +
                        descendant.substr(descendant.find(':'));
                    result.push_back(newDescendant);
                }
            }
        }
    }

    return result;
}
```

Hint: findAscendant function will have a very similar approach.

Outputs:

The program prints:

- The menu options.
- The result of the selected action:

For option 1:

- Person name
- Birth year
- Parents (if any)

- Children (if any)

For option 2:

- List of ancestors by generation

For option 3:

- List of descendants by generation

For option 4:

- Confirmation of person added

For option 5:

- Confirmation of relationship established

For option 6:

- "Goodbye!" when exiting.

Implementation Requirements:

- Use recursion for the findAncestors, findDescendants functions
- The Person.h and FamilyTree.h files have been shared with you. You are required to implement the class definitions from these headers and submit the corresponding **Person.cpp** and **FamilyTree.cpp** files. These .cpp files must be compatible with the given header files.
- Handle file I/O errors and user input validation
- Follow the specified menu format

About Submission!

Homework will not be submitted automatically. Students are responsible for manually submitting their assignments on SUCourse before the deadline. Failure to submit will result in a zero grade

IMPORTANT!

If your code does not compile, then you will get **zero**. Please be careful about this and double check your code before submission.

Note: Please avoid using cin.get(), cin.ignore() in your code for your assignment. We are using CodeRunner, and these functions may cause unexpected behaviour.

VERY IMPORTANT!

Your programs will be compiled, executed and evaluated automatically; therefore you should definitely follow the rules for prompts, inputs and outputs. You can check the example test case outputs from SUCourse to get more information about the expected output.

Order of inputs and outputs must be in the mentioned format.

Following these rules is crucial for grading, otherwise, our software will not be able to process your outputs and you will lose some points in the best scenario.

Sample Run

Below, we provide only one sample run of the program that you will develop, for more sample runs please check the SUCourse example test cases.

The *italic* and **bold** phrases are inputs taken from the user.
NOTE THAT these inputs and the newlines after the inputs are missing at SUCourse in the outputs expected from you, so please ignore this as you copy/paste your C++ code from VS/XCode to SUCourse, the same will happen to your code too. You can see samples of SuCourse output in your assignment on SuCourse.

Visual Studio/XCode Outputs

Sample Run 1

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
```

```
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 1
Enter person name: Richard_Smith
Name: Richard_Smith
Birth Year: 1950
Parents: George_Smith (father), Dorothy_Smith (mother)
Children: Thomas_Smith

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 2

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 1
```

```
Enter person name: NonExistingPerson
Person not found: NonExistingPerson

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 3

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 2
Enter person name: Jacob
Enter number of generations to search: 2
Ancestors of Jacob (up to 2 generations):
Generation 1: Rebecca
Generation 1: Isaac
Generation 2: Sarah
Generation 2: Abraham

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 4

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 4
Enter name: Ayse
Enter birth year: 1960

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 1
Enter person name: Ayse
Name: Ayse
Birth Year: 1960
Parents: None
Children: None

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 5
Enter parent name: Fatma
Is this parent a mother (m) or father (f)? m
Enter child name (or type '-' for no child): Ayse
Person not found: Fatma
Failed to add relationship.

Menu:
```

```
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 5

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 4
Enter name: Adam
Enter birth year: 1999
Person Adam already exists.

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 6

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!
```

```
Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 3
Enter person name: GenerationVI_2
Enter number of generations to search: 2
Descendants of GenerationVI_2 (up to 2 generations):
No known descendants for GenerationVI_2.

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 7

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 2
Enter person name: Joseph
Enter number of generations to search:  Ancestors of Joseph (up
to 0 generations):0
No known ancestors for Joseph.
```

```
Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

Sample Run 8

```
Enter people file name: people.txt
Enter relationships file name: relationships.txt
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 4
Enter name: Ayse
Enter birth year: 1960

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 1
Enter person name: Ayse
Name: Ayse
Birth Year: 1960
Parents: None
Children: None

Menu:
```

1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: ***4***
Enter name: ***Fatma***
Enter birth year: ***1942***

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: ***5***
Enter parent name: ***Fatma***
Is this parent a mother (m) or father (f)? ***m***
Enter child name (or type '-' for no child): ***Ayse***
Relationship added successfully.

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: ***1***
Enter person name: ***Ayse***
Name: Ayse
Birth Year: 1960
Parents: Fatma (mother)
Children: None

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person

5. Add a parent-child relationship
6. Exit
Enter choice: ***6***
Goodbye!

Sample Run 9

Enter people file name: ***people.txt***
Enter relationships file name: ***relationships.txt***
Family tree loaded successfully!

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: ***2***
Enter person name: ***Joseph***
Enter number of generations to search: ***1***
Ancestors of Joseph (up to 1 generations):
Generation 1: Rachel
Generation 1: Jacob

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: ***3***
Enter person name: ***Joseph***
Enter number of generations to search: ***0***
Descendants of Joseph (up to 0 generations):
No known descendants for Joseph.

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person

```
5. Add a parent-child relationship
6. Exit
Enter choice: 3
Enter person name: Joseph
Enter number of generations to search: 1
Descendants of Joseph (up to 1 generations):
Generation 1: Ephraim, Manasseh, Joshua

Menu:
1. Display a person's information
2. Find ancestors of a person
3. Find descendants of a person
4. Add a new person
5. Add a parent-child relationship
6. Exit
Enter choice: 6
Goodbye!
```

General Rules and Guidelines about Homework

The following rules and guidelines will be applicable to all homework unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) or LAs (Learning Assistants) of CS201. Office hours of TAs/LAs are at the SUCourse.

What and Where to Submit

You can prepare (or at least test) your program using MS Visual Studio 2022 C++ (Windows users) or using XCode (macOS users).

- Your code will be automatically graded using SUCourse. Therefore, it is essential that you ensure your output matches the exact same outputs given in the example test cases provided by SUCourse.
- After writing your code, use the "Check" button located under the code editor in SUCourse to see your grade based on the example test cases used. This grade will give you an idea of how well your code is performing.
- Note that the example test cases used for checking your code are not the same as the ones used for grading your homework. Your final grade will be based on different test cases. Therefore, it is important that you carefully follow the instructions and ensure that your code is working correctly to achieve the best possible grade on your homework assignment.
- To submit your homework, click on the "Finish attempt..." button and then the "Submit all and finish" button. If you wish to submit again before the due date, you can press the "Re-attempt quiz" button.
- Submit your work **through SUCourse only**! You will receive no credits if you submit by any other means (email, paper, etc.).

Grading, Review and Objections

Be careful about the automatic grading: Your programs will be graded using an automated system. Therefore, you should follow the guidelines on the input and output order. Moreover, It is important to use the exact same text as

provided in the example test case outputs from SUCourse. Otherwise, the automated grading process will fail for your homework, and you may get a zero, or in the best scenario, you will lose points.

Grading:

- Late penalty is 10% of full grade (only 1 late day is allowed)
- Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.
- If your code does not work because of a syntax error, then we cannot grade it; and thus, your grade will be 0.
- Please submit your **own** work only. It is really easy to find "similar" programs!
- Plagiarism will not be tolerated. Please check our plagiarism policy given in the Syllabus.

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

Good Luck!
Ömer A. Teryaki & CS201 Instructors