Université de Limoges

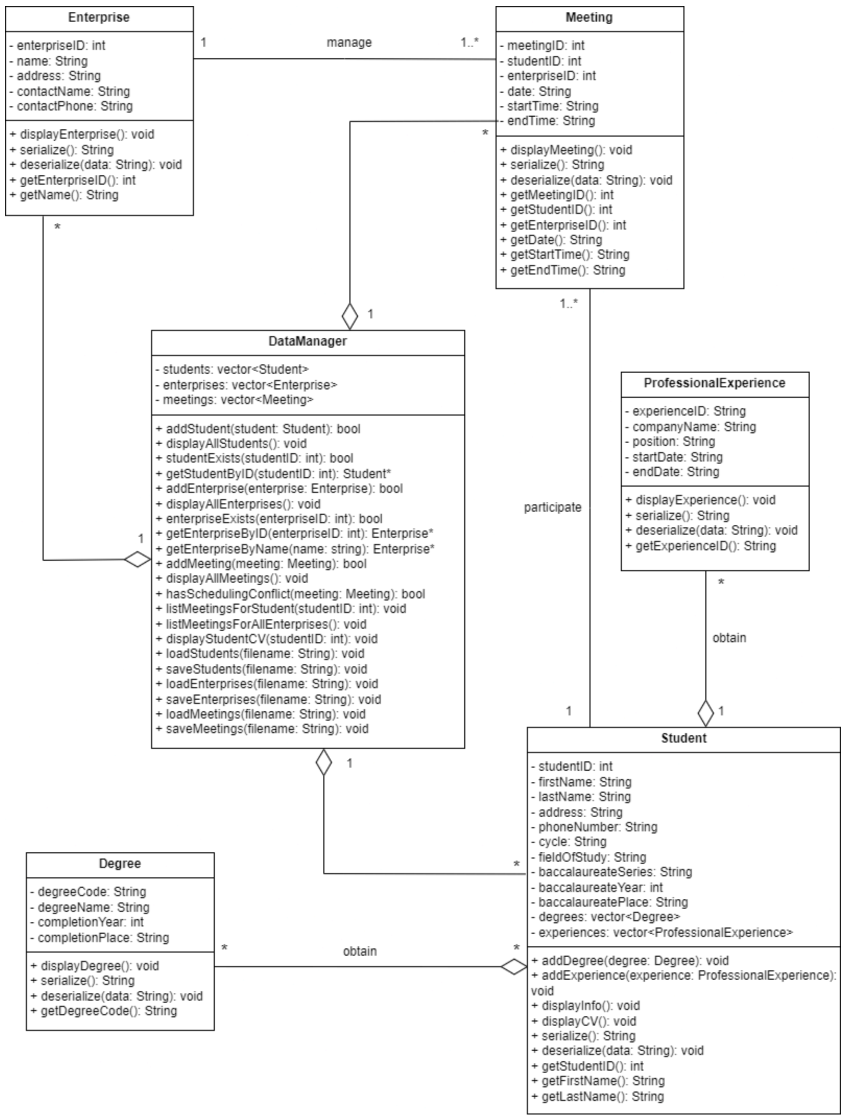
Report for APOO/C++ Project:

Student - Enterprise Meetings

Nom et Prénom: Viet HOANG
Quang Ha TRAN

Table of contents:

UML Class Diagram


Enterprise
- enterpriseID: int
- name: String
- address: String
- contactName: String
- contactPhone: String
+ displayEnterprise(): void
+ serialize(): String
+ deserialize(data: String): void
+ getEnterpriseID(): int
+ getName(): String
1
manage
1..*
Meeting
- meetingID: int
- studentID: int
- enterpriseID: int
- date: String
- startTime: String
- endTime: String
+ displayMeeting(): void
+ serialize(): String
+ deserialize(data: String): void
+ getMeetingID(): int
+ getStudentID(): int
+ getEnterpriseID(): int
+ getDate(): String
+ getStartTime(): String
+ getEndTime(): String
*
*
1
1..*
DataManager
- students: vector<Student>
- enterprises: vector<Enterprise>
- meetings: vector<Meeting>
+ addStudent(student: Student): bool
+ displayAllStudents(): void
+ studentExists(studentID: int): bool
+ getStudentByID(studentID: int): Student*
+ addEnterprise(enterprise: Enterprise): bool
+ displayAllEnterprises(): void
+ enterpriseExists(enterpriseID: int): bool
+ getEnterpriseByID(enterpriseID: int): Enterprise*
+ getEnterpriseByName(name: string): Enterprise*
+ addMeeting(meeting: Meeting): bool
+ displayAllMeetings(): void
+ hasSchedulingConflict(meeting: Meeting): bool
+ listMeetingsForStudent(studentID: int): void
+ listMeetingsForAllEnterprises(): void
+ displayStudentCV(studentID: int): void
+ loadStudents(filename: String): void
+ saveStudents(filename: String): void
+ loadEnterprises(filename: String): void
+ saveEnterprises(filename: String): void
+ loadMeetings(filename: String): void
+ saveMeetings(filename: String): void
1
ProfessionalExperience
- experienceID: String
- companyName: String
- position: String
- startDate: String
- endDate: String
+ displayExperience(): void
+ serialize(): String
+ deserialize(data: String): void
+ getExperienceID(): String
participate
*
obtain
1
1
1
Student
- studentID: int
- firstName: String
- lastName: String
- address: String
- phoneNumber: String
- cycle: String
- fieldOfStudy: String
- baccalaureateSeries: String
- baccalaureateYear: int
- baccalaureatePlace: String
- degrees: vector<Degree>
- experiences: vector<ProfessionalExperience>
+ addDegree(degree: Degree): void
+ addExperience(experience: ProfessionalExperience): void
+ displayInfo(): void
+ displayCV(): void
+ serialize(): String
+ deserialize(data: String): void
+ getStudentID(): int
+ getFirstName(): String
+ getLastName(): String
*
Degree
- degreeCode: String
- degreeName: String
- completionYear: int
- completionPlace: String
+ displayDegree(): void
+ serialize(): String
+ deserialize(data: String): void
+ getDegreeCode(): String
*
obtain
*

Explanation of modeling choices for the UML Class Diagram:

1. **Classes:**
   - **1.1. Student class**
     - It is used to store personal information as well as educational and professional details.
     - Depending on which cycle the student is (First–cycle or Second-cycle), the class stores information of each type of student accordingly.
   - **1.2. Degree and ProfessionalExperience classes**
     - These classes represent the academic achievements and work experiences linked to the students.
   - **1.3. Enterprise class**

- This class handles the information of enterprises participating in meetings.

1.4. Meeting class

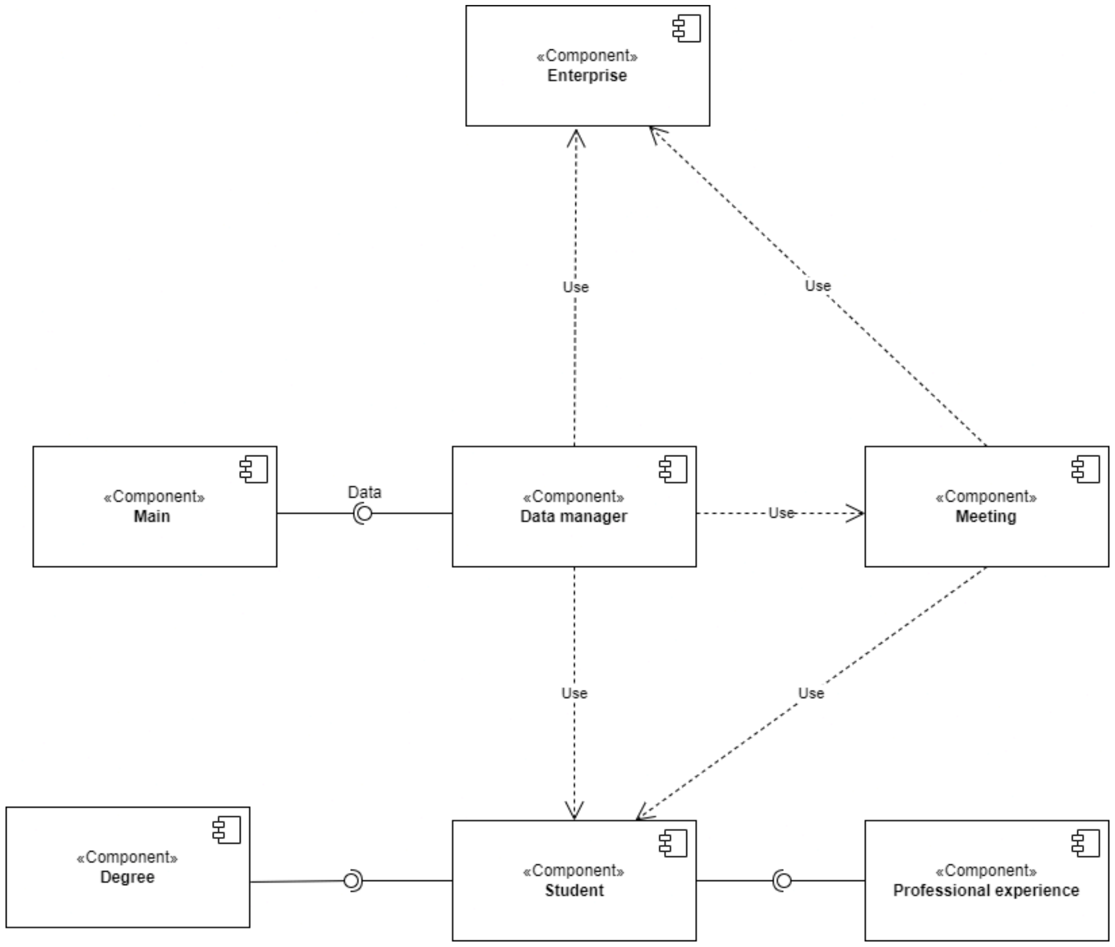
- This class represents the meetings where students and enterprises participate in.

1.5. DataManager class

- This class acts as a central controller for managing the Student, Enterprise and Meeting objects (adding, retrieving, displaying data,...).

2. Relationships

- Student - Degree: Student class aggregates Degree class with * to * multiplicity (a student can obtain multiple degrees and a degree can be obtained by multiple students) as the Degree objects are part of the student's information but can exist independently.
- Student - ProfessionalExperience:Student class aggregates ProfessionalExperience class with 1 to * multiplicity (a student can have multiple experience) as the ProfessionExperience objects are part of the student's information but can exist independently.
- Student - Meeting: Student class associates with Meeting class with 1 to 1..* multiplicity as a student can participate in 1 or many meetings.
- Enterprise - Meeting: Enterprise and Meeting have a 1..* association indicating that an enterprise can organize 1 or multiple meetings.



Component diagram

Objectives

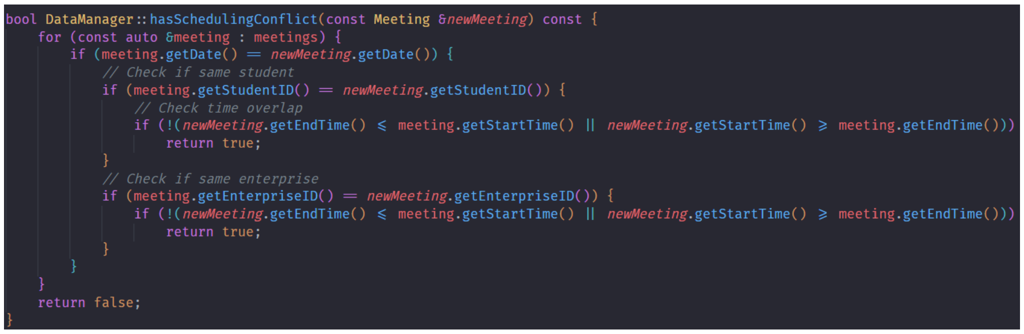
- **Manage Entities:** Efficiently add, store, and retrieve information about students, enterprises, and meetings.
- **Ensure Data Integrity:** Prevent duplication through unique identifiers and avoid scheduling conflicts.
- **Provide Comprehensive Views:** Allow users to display detailed information, including students' CVs and sorted meeting schedules.

Code elements

1. **Some specific algorithms:**

1.1. Detect scheduling conflict

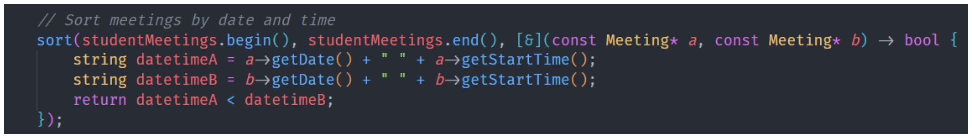
- Implement a method that can prevent overlapping meetings for students or enterprises on the same date.
- Since this project is a mini application of the Student-Enterprise Meetings system, the dataset is not too large and hence manageable by iterating through existing meetings and using conditional logic to check if time overlaps.



```cpp
bool DataManager::hasSchedulingConflict(const Meeting &newMeeting) const {
    for (const auto &meeting : meetings) {
        if (meeting.getDate() == newMeeting.getDate()) {
            // Check if same student
            if (meeting.getStudentID() == newMeeting.getStudentID()) {
                // Check time overlap
                if (!(newMeeting.getEndTime() <= meeting.getStartTime() || newMeeting.getStartTime() >= meeting.getEndTime()))
                    return true;
            }
            // Check if same enterprise
            if (meeting.getEnterpriseID() == newMeeting.getEnterpriseID()) {
                if (!(newMeeting.getEndTime() <= meeting.getStartTime() || newMeeting.getStartTime() >= meeting.getEndTime()))
                    return true;
            }
        }
    }
    return false;
}
```

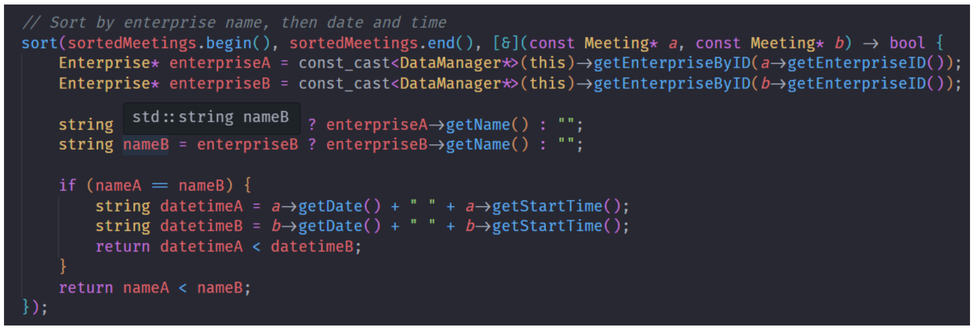
1.2. Sorting method

- Use sort() to produce sorted listings of meetings for students and enterprises because it is already an efficient built-in function of C++ and it also supports custom sorting logic.
- Sorting Logic: Combines the date and startTime of each meeting into a single string ("YYYY-MM-DD HH:MM") to leverage lexicographical ordering and ensure meetings are sorted chronologically.
- We implement the sort() method based on the objects:
    - Meetings for a specific student: sort by date and time



```cpp
// Sort meetings by date and time
sort(studentMeetings.begin(), studentMeetings.end(), [&](const Meeting* a, const Meeting* b) -> bool {
    string datetimeA = a->getDate() + " " + a->getStartTime();
    string datetimeB = b->getDate() + " " + b->getStartTime();
    return datetimeA < datetimeB;
});
```

- Meetings for all enterprises: sort by name first, then sort by date and time



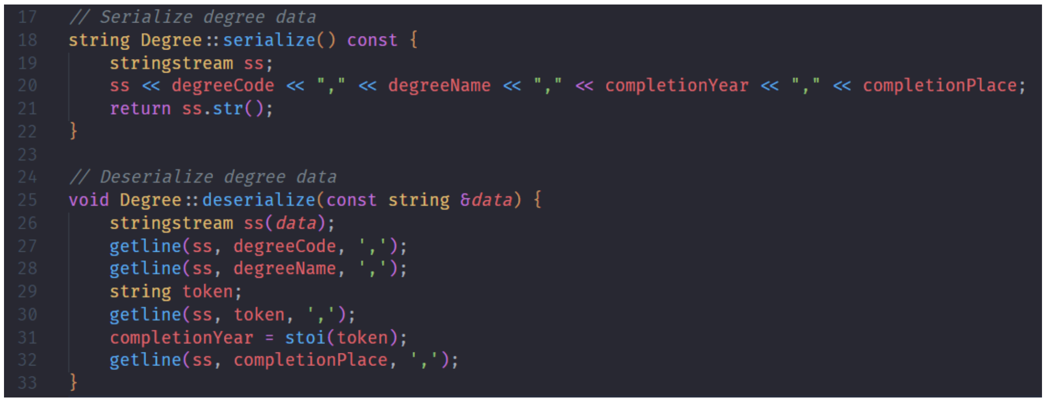
```cpp
// Sort by enterprise name, then date and time
sort(sortedMeetings.begin(), sortedMeetings.end(), [&](const Meeting* a, const Meeting* b) -> bool {
    Enterprise* enterpriseA = const_cast<DataManager*>(this)->getEnterpriseByID(a->getEnterpriseID());
    Enterprise* enterpriseB = const_cast<DataManager*>(this)->getEnterpriseByID(b->getEnterpriseID());

    string  std::string nameB  ? enterpriseA->getName() : "";
    string nameB = enterpriseB ? enterpriseB->getName() : "";

    if (nameA == nameB) {
        string datetimeA = a->getDate() + " " + a->getStartTime();
        string datetimeB = b->getDate() + " " + b->getStartTime();
        return datetimeA < datetimeB;
    }
    return nameA < nameB;
});
```

1.3. Serialization and Deserialization

- In addition to adding students/enterprises/meetings manually, we create methods to take a text file as input and load them to the system. Plus, when adding a new student/enterprise/meeting to the system, we can also save it into our respective text files.
- Implementation:
  - For each class above, we implement both serialize() (convert object to string) and deserialize() (convert string to object) methods to use in the loading and saving processes.
  - We use stringstream (a string manipulation utility) for structured string parsing, with a consistent delimiter (|) for easy parsing during deserialization.

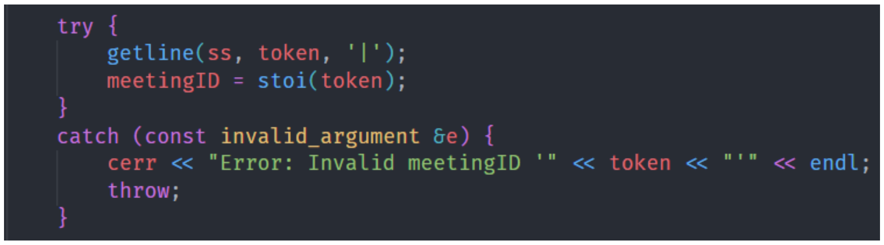


```cpp
// Serialize degree data
string Degree::serialize() const {
    stringstream ss;
    ss << degreeCode << "," << degreeName << "," << completionYear << "," << completionPlace;
    return ss.str();
}

// Deserialize degree data
void Degree::deserialize(const string &data) {
    stringstream ss(data);
    getline(ss, degreeCode, ',');
    getline(ss, degreeName, ',');
    string token;
    getline(ss, token, ',');
    completionYear = stoi(token);
    getline(ss, completionPlace, ',');
}
```

1.4. Deserialization validation

- Utilizes getline with the delimiter | to parse individual data fields sequentially.
- Error Handling: Employs stoi within try-catch blocks to safely convert string tokens to integers, preventing runtime errors due to invalid input.
- Hierarchical Parsing: Recognizes section markers (DEGREES_START, EXPERIENCES_START) to appropriately parse and instantiate nested objects (Degree, ProfessionalExperience).

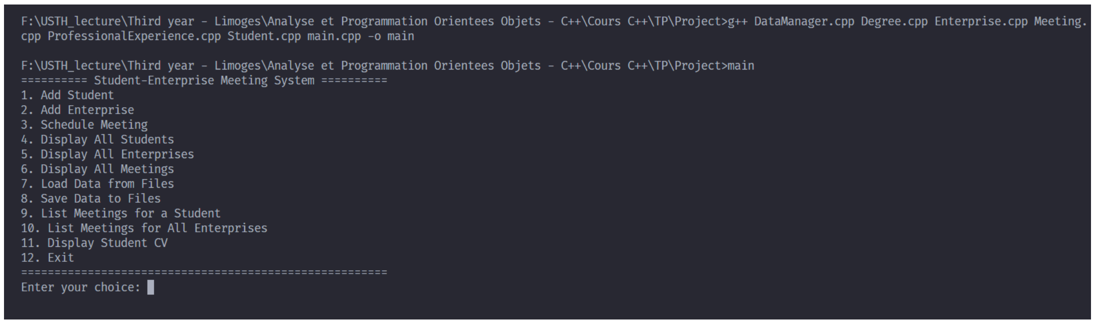


```cpp
try {
    getline(ss, token, '|');
    meetingID = stoi(token);
}
catch (const invalid_argument &e) {
    cerr << "Error: Invalid meetingID '" << token << "'" << endl;
    throw;
}
```

2. **Implementation choices**
   - **2.1. Use of container**
     - For this project, we use the vector container for storing the Student, Enterprise and Meeting objects because vector supports dynamic resizing and is flexible for varying dataset sizes. It also provides efficient iteration for adding and traversing elements.
   - **2.2. Modular design**
     - We create a specific class, DataManager, to act as a controller class for managing collections (students, enterprises, meetings) and dealing with specific functionalities (display sorted lists, display CV,...).
     - Although list allows for efficient insertions and deletions, it lacks cache locality and does not support random access, making it less suitable for this application.
     - set or unordered_set could enforce uniqueness but would complicate serialization and ordering.

Code execution

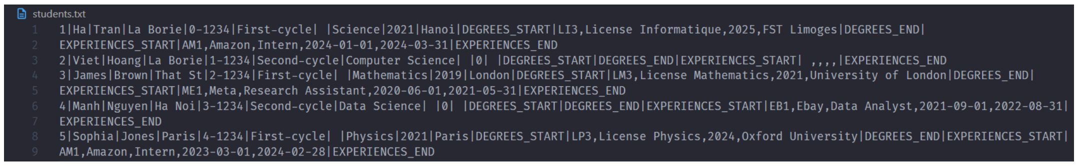
- First, compile the codes:
  g++ DataManager.cpp Degree.cpp Enterprise.cpp Meeting.cpp ProfessionalExperience.cpp Student.cpp main.cpp -o main
- Then run: main



```
F:\USTH_lecture\Third year - Limoges\Analyse et Programmation Orientees Objets - C++\Cours C++\TP\Project>g++ DataManager.cpp Degree.cpp Enterprise.cpp Meeting.
cpp ProfessionalExperience.cpp Student.cpp main.cpp -o main

F:\USTH_lecture\Third year - Limoges\Analyse et Programmation Orientees Objets - C++\Cours C++\TP\Project>main
========== Student-Enterprise Meeting System ==========
1. Add Student
2. Add Enterprise
3. Schedule Meeting
4. Display All Students
5. Display All Enterprises
6. Display All Meetings
7. Load Data from Files
8. Save Data to Files
9. List Meetings for a Student
10. List Meetings for All Enterprises
11. Display Student CV
12. Exit
=======================================================
Enter your choice: 
```

- From here, we can manually add students, enterprises and schedule the meetings, or we can write a text file containing the information we need and load them to the system. Here are the example files to load:
  - *students.txt*

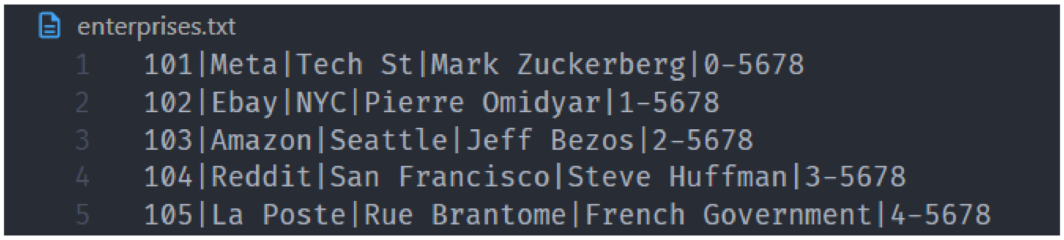


students.txt

```
1|Ha|Tran|La Borie|0-1234|First-cycle| |Science|2021|Hanoi|DEGREES_START|LI3,License Informatique,2025,FST Limoges|DEGREES_END|
EXPERIENCES_START|AM1,Amazon,Intern,2024-01-01,2024-03-31|EXPERIENCES_END
2|Viet|Hoang|La Borie|1-1234|Second-cycle|Computer Science| |0| |DEGREES_START|DEGREES_END|EXPERIENCES_START| ,,,,|EXPERIENCES_END
3|James|Brown|That St|2-1234|First-cycle| |Mathematics|2019|London|DEGREES_START|LM3,License Mathematics,2021,University of London|DEGREES_END|
EXPERIENCES_START|ME1,Meta,Research Assistant,2020-06-01,2021-05-31|EXPERIENCES_END
4|Manh|Nguyen|Ha Noi|3-1234|Second-cycle|Data Science| |0| |DEGREES_START|DEGREES_END|EXPERIENCES_START|EB1,Ebay,Data Analyst,2021-09-01,2022-08-31|
EXPERIENCES_END
5|Sophia|Jones|Paris|4-1234|First-cycle| |Physics|2021|Paris|DEGREES_START|LP3,License Physics,2024,Oxford University|DEGREES_END|EXPERIENCES_START|
AM1,Amazon,Intern,2023-03-01,2024-02-28|EXPERIENCES_END
```

- The details are separated with '|' symbol
- Write the information in this order:
  - For the First-cycle student: *student_ID*|*first_name*|*last_name*|*address*|*phone_number*|*cycle*| |*baccalaureates*|*year_obtained*|*place_obtained*|DEGREES_START|*degree_code, name, completion_year, completion_place*|DEGREES_END|EXPERIENCE_START|*exp_ID, company_name, position, start_date, end_date*|EXPERIENCE_END (here, the DEGREES_START, DEGREES_END, EXPERIENCE_START, EXPERIENCE_END act as markers to start/stop deserialization process)
    - For the Second-cycle student: change the |*baccalaureates*|*year_obtained*|*place_obtained*| part into *field_of_study*| |0| (the |0| indicates that for second-cycle students, we do not need to manage his/her baccalaureates so we initialize it as 0)

- *enterprises.txt*

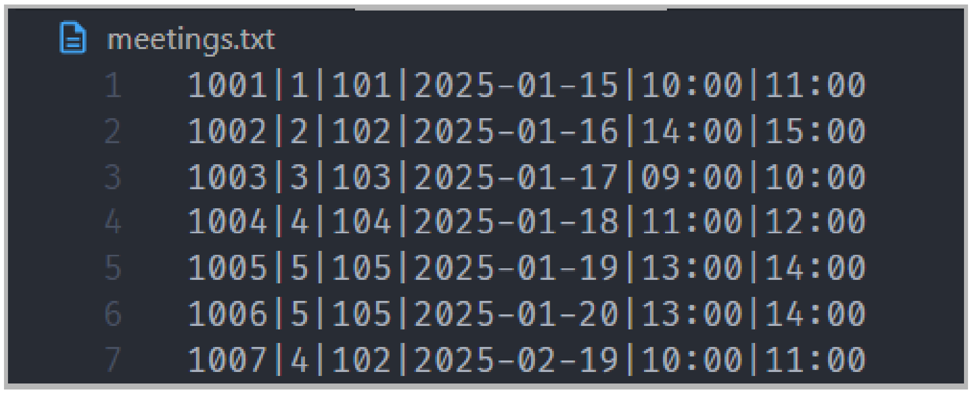


enterprises.txt

```
101|Meta|Tech St|Mark Zuckerberg|0-5678
102|Ebay|NYC|Pierre Omidyar|1-5678
103|Amazon|Seattle|Jeff Bezos|2-5678
104|Reddit|San Francisco|Steve Huffman|3-5678
105|La Poste|Rue Brantome|French Government|4-5678
```

Similar to the *students.txt*, the file is written in the order: *enterprise_ID*|*name*|*address*|*contact_name*|*contact_number*

- *meetings.txt*



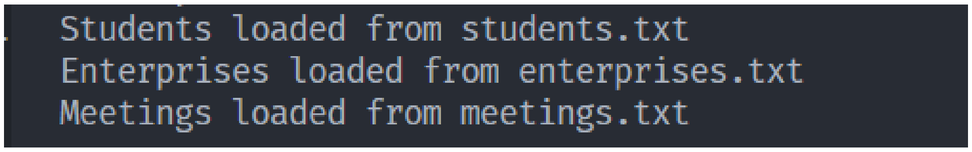
meetings.txt

```
1001|1|101|2025-01-15|10:00|11:00
1002|2|102|2025-01-16|14:00|15:00
1003|3|103|2025-01-17|09:00|10:00
1004|4|104|2025-01-18|11:00|12:00
1005|5|105|2025-01-19|13:00|14:00
1006|5|105|2025-01-20|13:00|14:00
1007|4|102|2025-02-19|10:00|11:00
```

The file is written in this order:

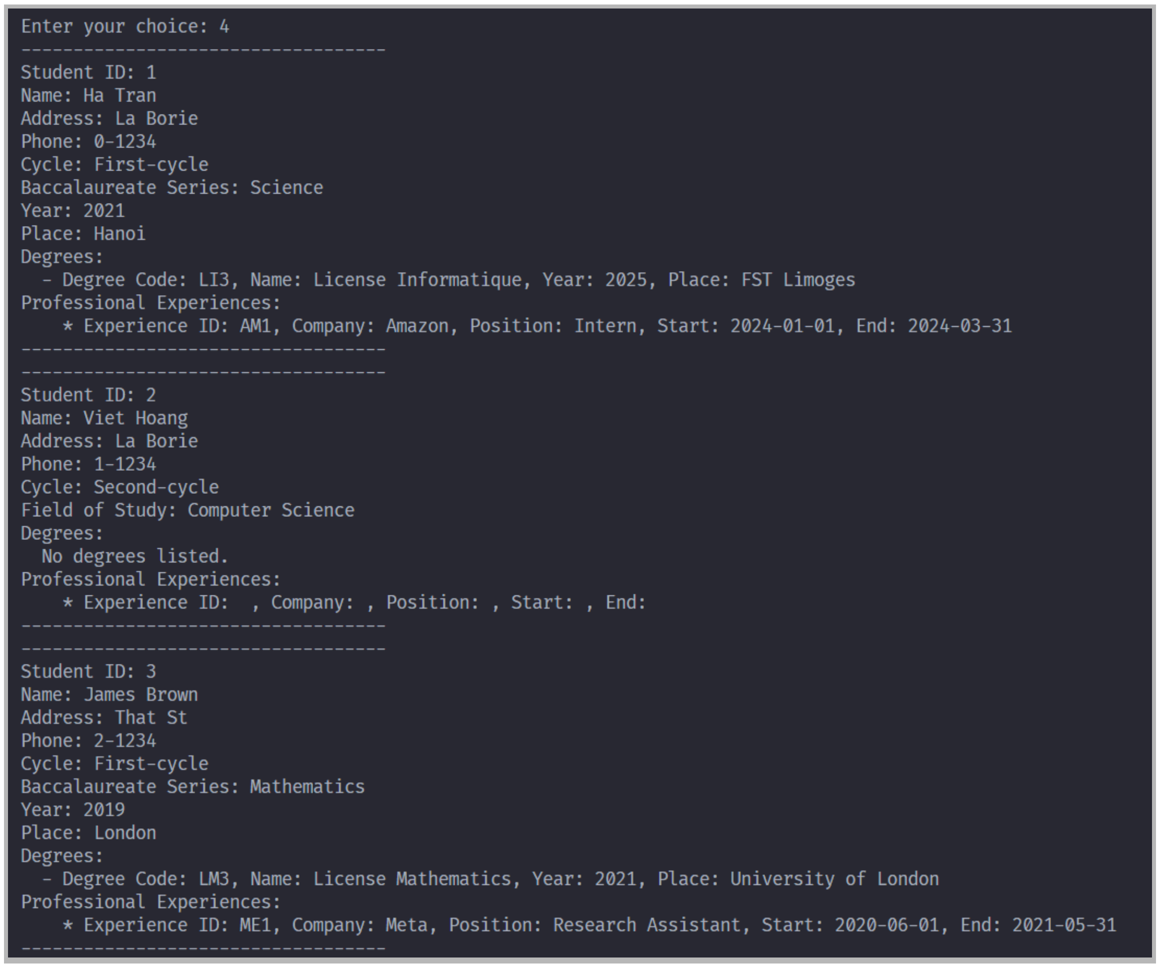
meeting_ID|student_ID|enterprise_ID|date|start_time|end_time

- Upon loading data from the above files, we will have the message:

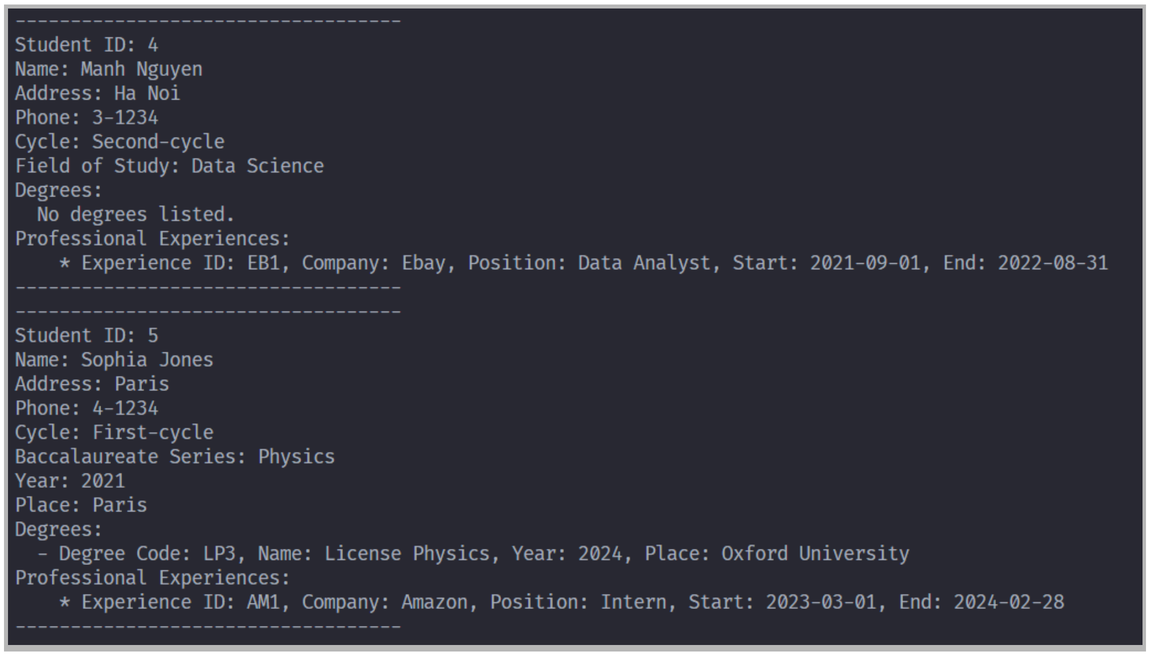


```
Students loaded from students.txt
Enterprises loaded from enterprises.txt
Meetings loaded from meetings.txt
```

- After that, we can display all the information of the students, enterprises or the meeting details depending on our choice. Below is the example of displaying all students information:

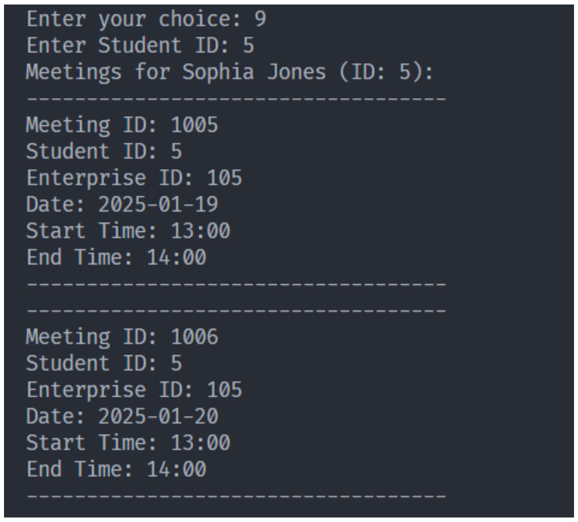


```
Enter your choice: 4
-----------------------------------
Student ID: 1
Name: Ha Tran
Address: La Borie
Phone: 0-1234
Cycle: First-cycle
Baccalaureate Series: Science
Year: 2021
Place: Hanoi
Degrees:
  - Degree Code: LI3, Name: License Informatique, Year: 2025, Place: FST Limoges
Professional Experiences:
    * Experience ID: AM1, Company: Amazon, Position: Intern, Start: 2024-01-01, End: 2024-03-31
-----------------------------------
-----------------------------------
Student ID: 2
Name: Viet Hoang
Address: La Borie
Phone: 1-1234
Cycle: Second-cycle
Field of Study: Computer Science
Degrees:
  No degrees listed.
Professional Experiences:
    * Experience ID:  , Company: , Position: , Start: , End:
-----------------------------------
-----------------------------------
Student ID: 3
Name: James Brown
Address: That St
Phone: 2-1234
Cycle: First-cycle
Baccalaureate Series: Mathematics
Year: 2019
Place: London
Degrees:
  - Degree Code: LM3, Name: License Mathematics, Year: 2021, Place: University of London
Professional Experiences:
    * Experience ID: ME1, Company: Meta, Position: Research Assistant, Start: 2020-06-01, End: 2021-05-31
-----------------------------------
```



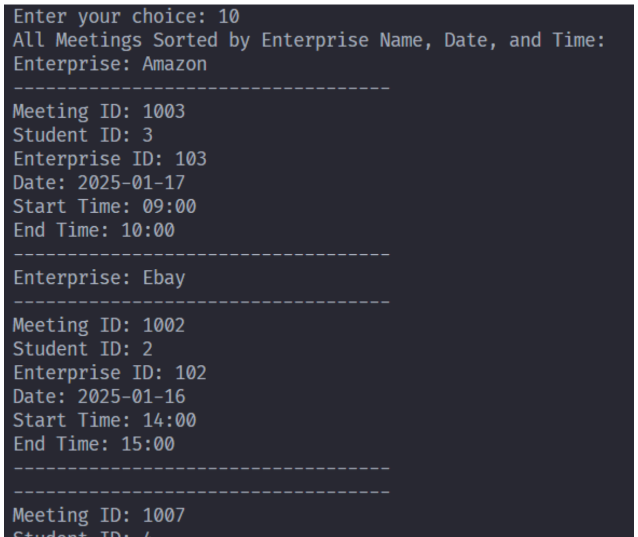
```
-----------------------------------
Student ID: 4
Name: Manh Nguyen
Address: Ha Noi
Phone: 3-1234
Cycle: Second-cycle
Field of Study: Data Science
Degrees:
  No degrees listed.
Professional Experiences:
    * Experience ID: EB1, Company: Ebay, Position: Data Analyst, Start: 2021-09-01, End: 2022-08-31
-----------------------------------
-----------------------------------
Student ID: 5
Name: Sophia Jones
Address: Paris
Phone: 4-1234
Cycle: First-cycle
Baccalaureate Series: Physics
Year: 2021
Place: Paris
Degrees:
  - Degree Code: LP3, Name: License Physics, Year: 2024, Place: Oxford University
Professional Experiences:
    * Experience ID: AM1, Company: Amazon, Position: Intern, Start: 2023-03-01, End: 2024-02-28
-----------------------------------
```

- We can list all the appointments for a specific student sorted using date and time. For example, below I list meetings for student with ID 5:

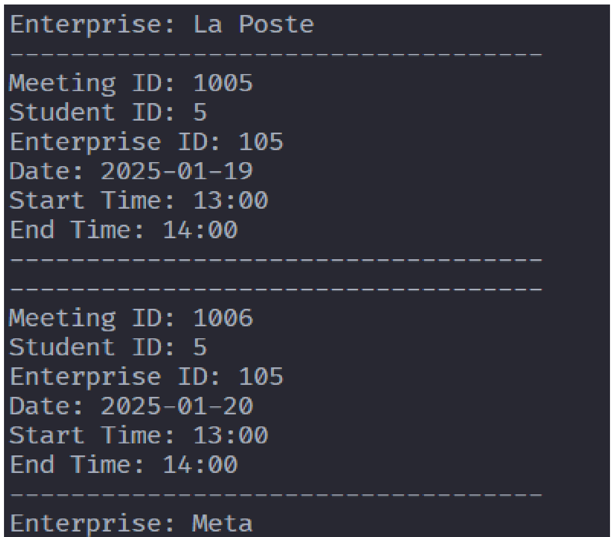


```
Enter your choice: 9
Enter Student ID: 5
Meetings for Sophia Jones (ID: 5):
-----------------------------------
Meeting ID: 1005
Student ID: 5
Enterprise ID: 105
Date: 2025-01-19
Start Time: 13:00
End Time: 14:00
-----------------------------------
-----------------------------------
Meeting ID: 1006
Student ID: 5
Enterprise ID: 105
Date: 2025-01-20
Start Time: 13:00
End Time: 14:00
-----------------------------------
```

- We can also list all the meetings for all companies giving appointments (sorted by company name and then by date and time):

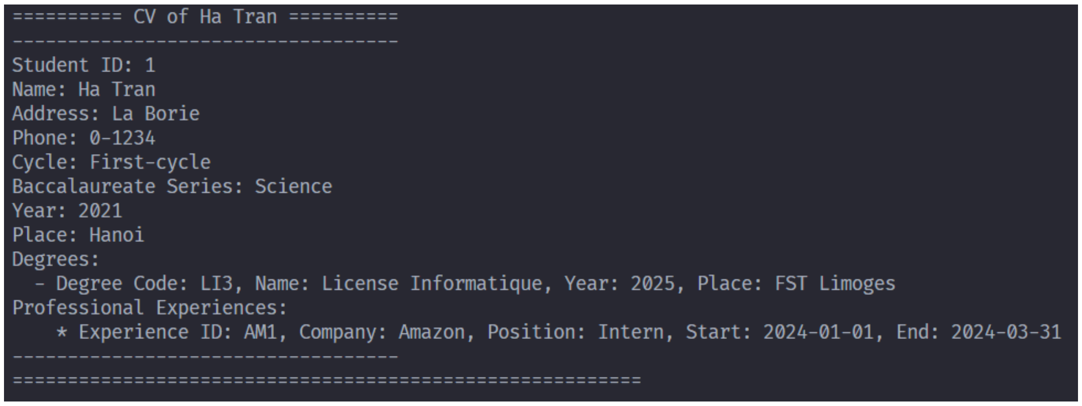


```
Enter your choice: 10
All Meetings Sorted by Enterprise Name, Date, and Time:
Enterprise: Amazon
-----------------------------------
Meeting ID: 1003
Student ID: 3
Enterprise ID: 103
Date: 2025-01-17
Start Time: 09:00
End Time: 10:00
-----------------------------------
Enterprise: Ebay
-----------------------------------
Meeting ID: 1002
Student ID: 2
Enterprise ID: 102
Date: 2025-01-16
Start Time: 14:00
End Time: 15:00
-----------------------------------
-----------------------------------
Meeting ID: 1007
```



```
Enterprise: La Poste
-----------------------------------
Meeting ID: 1005
Student ID: 5
Enterprise ID: 105
Date: 2025-01-19
Start Time: 13:00
End Time: 14:00
-----------------------------------
-----------------------------------
Meeting ID: 1006
Student ID: 5
Enterprise ID: 105
Date: 2025-01-20
Start Time: 13:00
End Time: 14:00
-----------------------------------
Enterprise: Meta
```

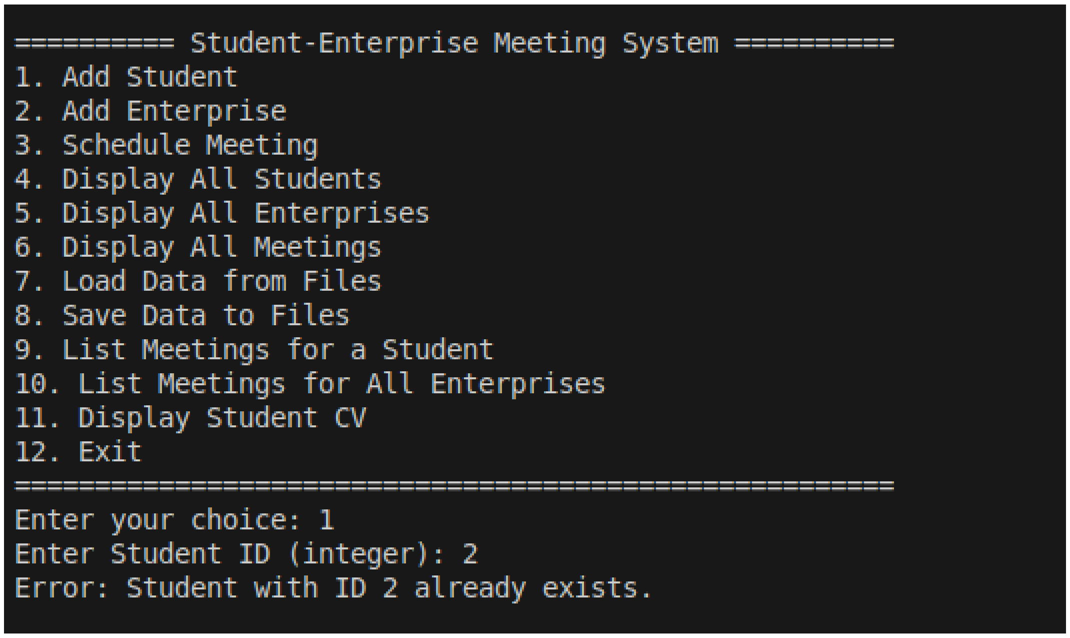
- Then, we can display the “CV” for a specific student:



```
========== CV of Ha Tran ==========
-----------------------------------
Student ID: 1
Name: Ha Tran
Address: La Borie
Phone: 0-1234
Cycle: First-cycle
Baccalaureate Series: Science
Year: 2021
Place: Hanoi
Degrees:
  - Degree Code: LI3, Name: License Informatique, Year: 2025, Place: FST Limoges
Professional Experiences:
    * Experience ID: AM1, Company: Amazon, Position: Intern, Start: 2024-01-01, End: 2024-03-31
-----------------------------------
========================================================
```

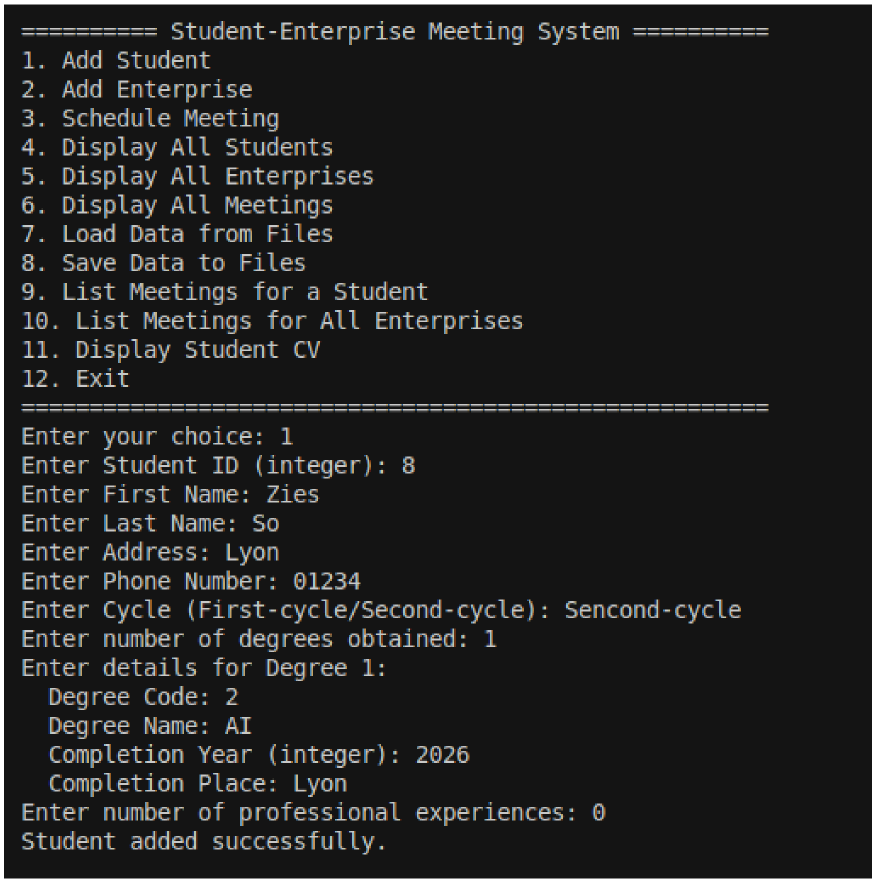
Manual adding

- We can also add a student directly by using option 1. For example:



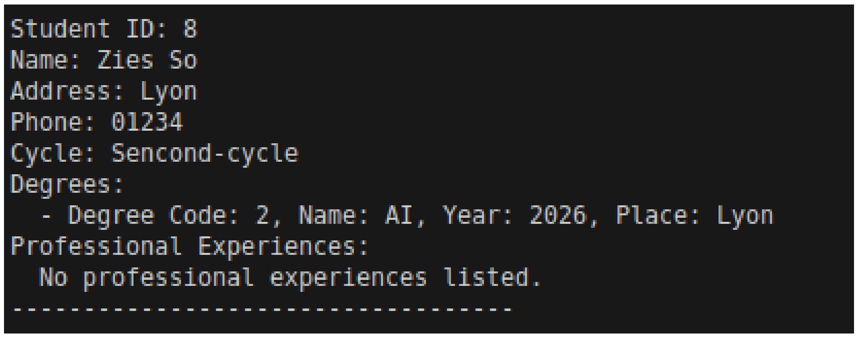
```
========== Student-Enterprise Meeting System ==========
1. Add Student
2. Add Enterprise
3. Schedule Meeting
4. Display All Students
5. Display All Enterprises
6. Display All Meetings
7. Load Data from Files
8. Save Data to Files
9. List Meetings for a Student
10. List Meetings for All Enterprises
11. Display Student CV
12. Exit
=======================================================
Enter your choice: 1
Enter Student ID (integer): 2
Error: Student with ID 2 already exists.
```

- The student with ID 2 already exists when we load from the file, therefore the program throws an error.
- So, now we change from ID 2 to ID 8, the program accepts and asks for additional details of the student (basic information, cycle, degree,...):



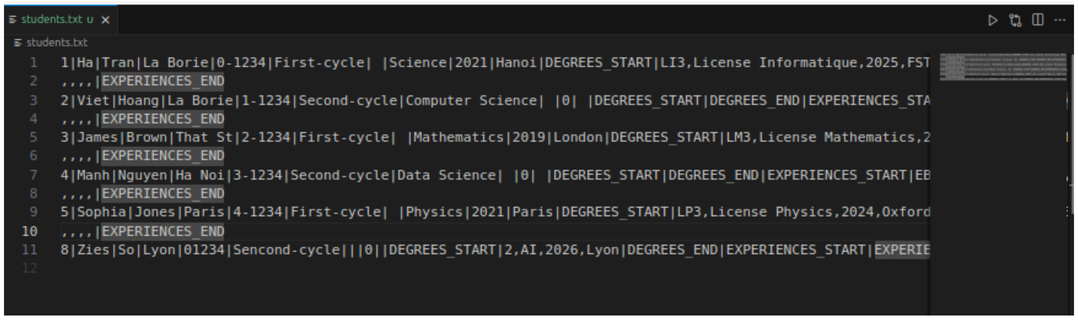
```
========== Student-Enterprise Meeting System ==========
1. Add Student
2. Add Enterprise
3. Schedule Meeting
4. Display All Students
5. Display All Enterprises
6. Display All Meetings
7. Load Data from Files
8. Save Data to Files
9. List Meetings for a Student
10. List Meetings for All Enterprises
11. Display Student CV
12. Exit
=======================================================
Enter your choice: 1
Enter Student ID (integer): 8
Enter First Name: Zies
Enter Last Name: So
Enter Address: Lyon
Enter Phone Number: 01234
Enter Cycle (First-cycle/Second-cycle): Sencond-cycle
Enter number of degrees obtained: 1
Enter details for Degree 1:
  Degree Code: 2
  Degree Name: AI
  Completion Year (integer): 2026
  Completion Place: Lyon
Enter number of professional experiences: 0
Student added successfully.
```

- The new student's information can be displayed using option 4:



```
Student ID: 8
Name: Zies So
Address: Lyon
Phone: 01234
Cycle: Sencond-cycle
Degrees:
  - Degree Code: 2, Name: AI, Year: 2026, Place: Lyon
Professional Experiences:
  No professional experiences listed.
-----------------------------------
```

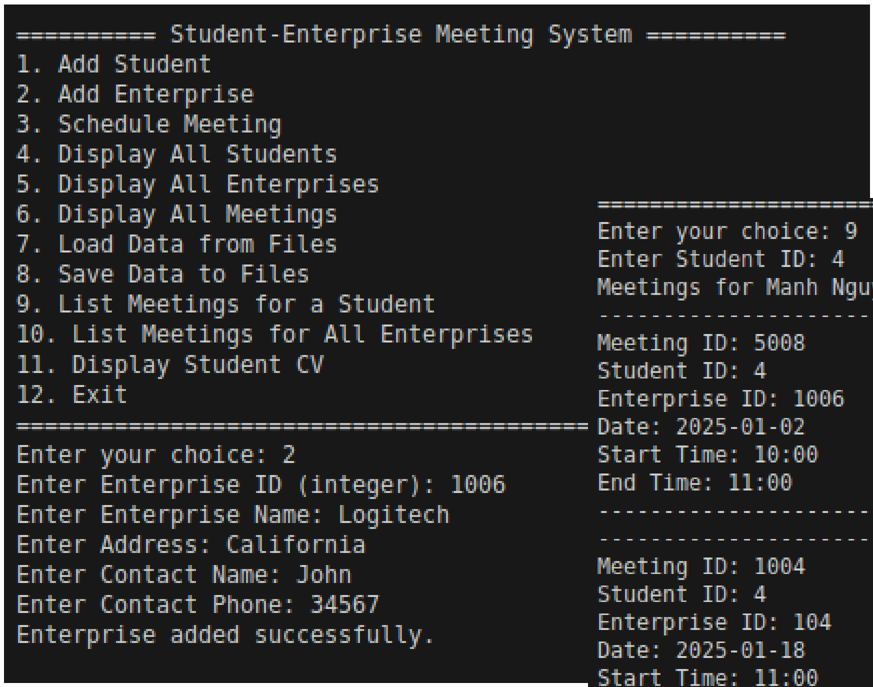
- We can also choose option 8 to save to file, after that, the *student.txt* file is added with the student ID8:



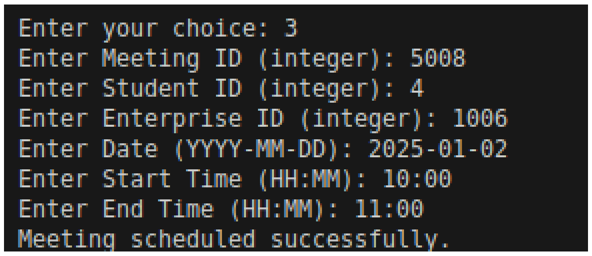
students.txt

```
1|Ha|Tran|La Borie|0-1234|First-cycle| |Science|2021|Hanoi|DEGREES_START|LI3,License Informatique,2025,FST
,,,,|EXPERIENCES_END
2|Viet|Hoang|La Borie|1-1234|Second-cycle|Computer Science| |0| |DEGREES_START|DEGREES_END|EXPERIENCES_STA
,,,,|EXPERIENCES_END
3|James|Brown|That St|2-1234|First-cycle| |Mathematics|2019|London|DEGREES_START|LM3,License Mathematics,2
,,,,|EXPERIENCES_END
4|Manh|Nguyen|Ha Noi|3-1234|Second-cycle|Data Science| |0| |DEGREES_START|DEGREES_END|EXPERIENCES_START|EB
,,,,|EXPERIENCES_END
5|Sophia|Jones|Paris|4-1234|First-cycle| |Physics|2021|Paris|DEGREES_START|LP3,License Physics,2024,Oxford
,,,,|EXPERIENCES_END
8|Zies|So|Lyon|01234|Sencond-cycle|||0||DEGREES_START|2,AI,2026,Lyon|DEGREES_END|EXPERIENCES_START|EXPERIE
```

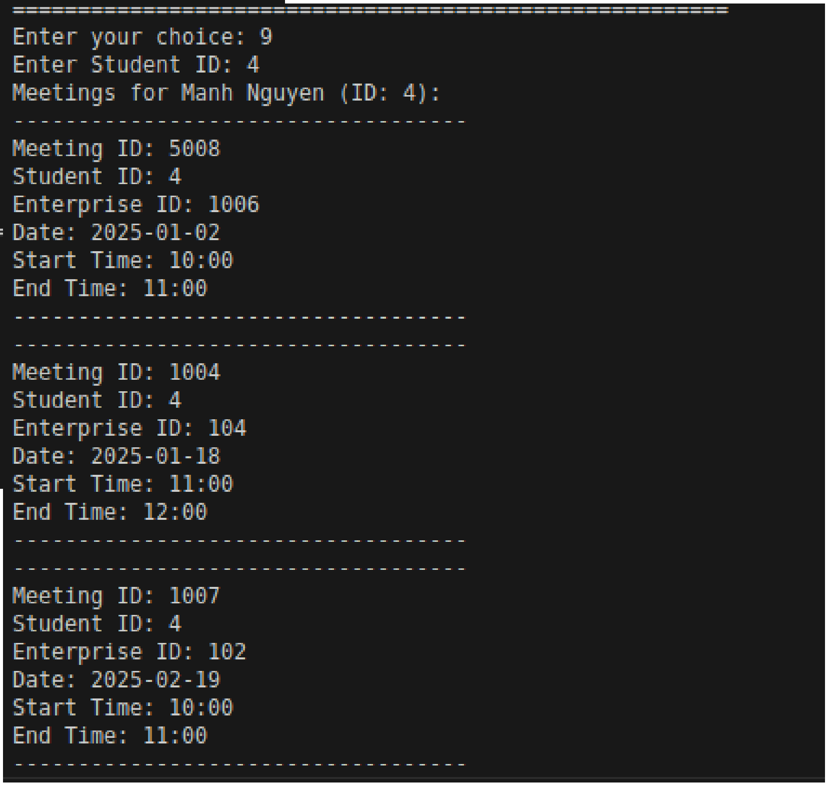
- Similar for adding Enterprise and Meeting:



```
=========== Student-Enterprise Meeting System ===========
1. Add Student
2. Add Enterprise
3. Schedule Meeting
4. Display All Students
5. Display All Enterprises
6. Display All Meetings
7. Load Data from Files
8. Save Data to Files
9. List Meetings for a Student
10. List Meetings for All Enterprises
11. Display Student CV
12. Exit
=========================================
Enter your choice: 2
Enter Enterprise ID (integer): 1006
Enter Enterprise Name: Logitech
Enter Address: California
Enter Contact Name: John
Enter Contact Phone: 34567
Enterprise added successfully.
```



```
Enter your choice: 3
Enter Meeting ID (integer): 5008
Enter Student ID (integer): 4
Enter Enterprise ID (integer): 1006
Enter Date (YYYY-MM-DD): 2025-01-02
Enter Start Time (HH:MM): 10:00
Enter End Time (HH:MM): 11:00
Meeting scheduled successfully.
```



```
====================================================
Enter your choice: 9
Enter Student ID: 4
Meetings for Manh Nguyen (ID: 4):
----------------------------------
Meeting ID: 5008
Student ID: 4
Enterprise ID: 1006
Date: 2025-01-02
Start Time: 10:00
End Time: 11:00
----------------------------------
----------------------------------
Meeting ID: 1004
Student ID: 4
Enterprise ID: 104
Date: 2025-01-18
Start Time: 11:00
End Time: 12:00
----------------------------------
----------------------------------
Meeting ID: 1007
Student ID: 4
Enterprise ID: 102
Date: 2025-02-19
Start Time: 10:00
End Time: 11:00
----------------------------------
```