Wedding Agency Organizer

Final Project Documentation

Aleksander Świniarski

1. Description of the project

1. General Description

Program was created with idea to simplify the work of professional wedding planners from wedding agencies. Organization of the single wedding is already a hard analytic and organizational task, let alone when we need to prepare them for a dozen of clients.

This program organizes a list of clients and the list of their weddings. Weddings are sorted by dates. In the clients list we have vital information about each of clients: personal details, contact details and the budget they are willing to spend on their wedding.

In the weddings list we can find crucial information about organisation of each wedding like date of the wedding, guests list, schedule, organizational list of locations and settings and list of notes. List of weddings and clients are connected, so we can easily find a wedding by its date or by the surname of the couple and check when each couple has got a wedding.

Guests list included in each of the weddings contains information about each of the guest name and surname and also about its food preference and confirmation status.

Organizational list is divided into two parts. First consists a list of locations with their: names, addresses, guest's capacity, cost for each guest and status of the reservation. Second part consists of wedding settings and their name, price and status of preparation.

Schedule enables to organize time periods of each wedding and describe what happens at which moment.

Notes list is special list where wedding planner can note clients' special information. Each of the note has their title so it is easy to find specific note.

Program allows a fast manipulation of each lists, and because of information contained in them, program can quickly check:

- If cost of wedding is not exceeding the budget
- What elements need to be prepared for wedding of a specific client
- If number of guests is not too big for reserved locations
- How many guests confirmed their presence
- Most expensive elements of given wedding
- Number of the dishes on specific wedding with specifications depending on guests' preferences
- Contents of each list
- When is the nearest wedding
- Which weddings are in given time period

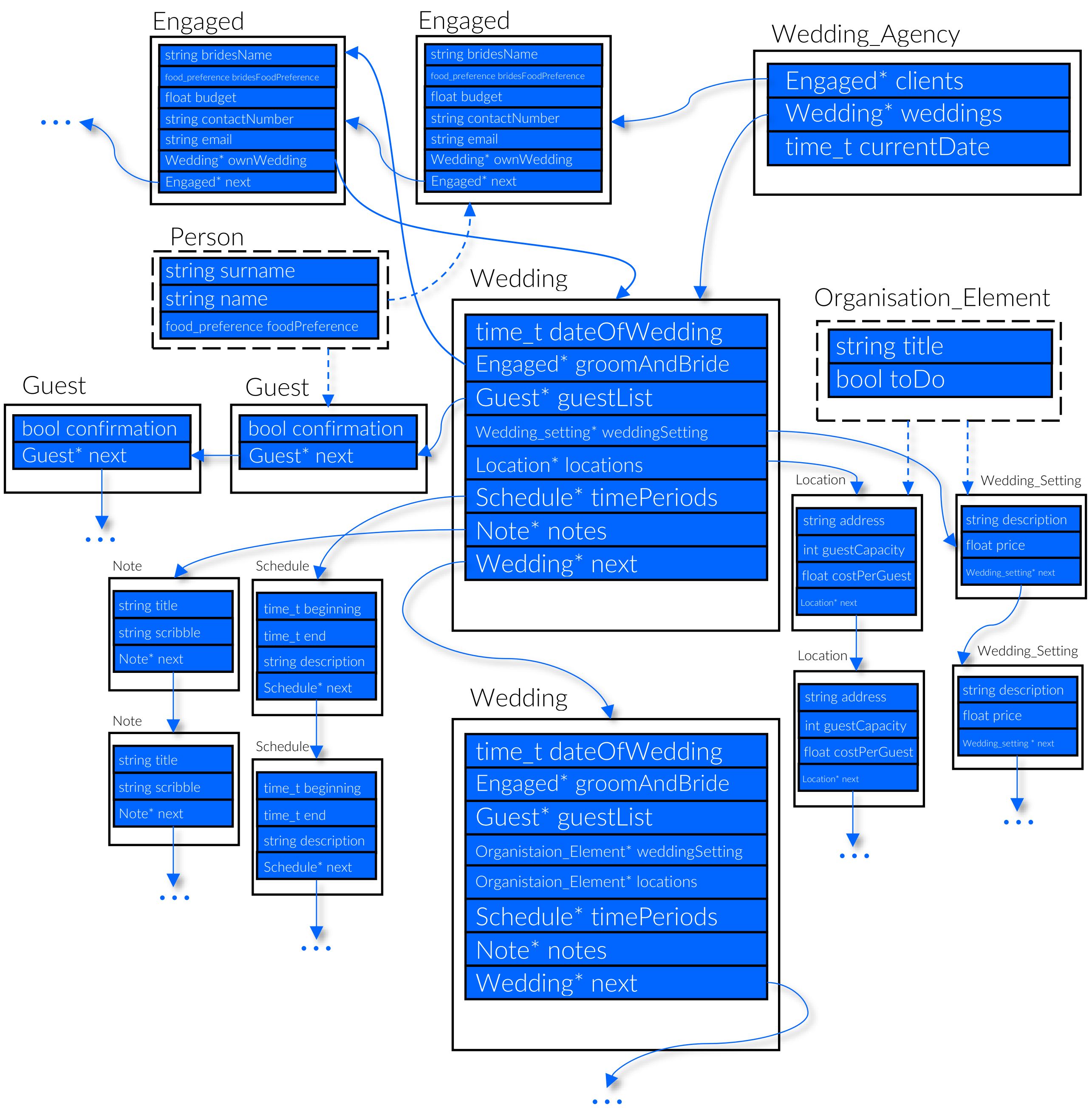
2. Restrictions

Program does not allow for following situations:

1. We cannot add a wedding with date before the current date
2. There cannot be more than one wedding on each date
3. The Client and the wedding must be in the correlation; they cannot point to the different elements.
4. In the guests' list there cannot be more than one person with the same name and surname
5. In the organizational list there cannot be more than two elements with the same title
6. In the notes list there cannot be multiple notes with the same title
7. In the schedule there cannot be multiple time periods with exactly the same time span, but they can overlap
8. If there are more than two couples with the same surname, we cannot find wedding by the couple's surname
9. Prices and budgets cannot be negative

2. Case study

1. Memory map



2. Overview of classes on map

The memory map above presents how objects in program are connected. Special elements of the diagram that are elements with lineolate perimeter are base classes and objects at the end of lineolate arrows are objects derived from this class. Arrows represent connection between objects with pointers.

The Wedding_Agency object is responsible for two lists and its contents. With such sorted data Wedding_Agency object can easily find specific list and summarise its contents, even sometimes compare data from different lists inside of Wedding object.

3. Declaration of Classes

1. Wedding Agency

```
    // Methods will have versions to find a wedding to modify by couples' surname or
    // date of wedding, if on the client list exists more than one couple with such
    // surname, program will display their info and dates of their weddings so user
    // can still find the wedding by the date

class Wedding_Agency
{

    Engaged* clients;
    Wedding* weddings;
    // These are pointers to main lists of the program

    time_t currentDate;
    // Date variable so program can check if the wedding has proper date

private:

    void removeAllElements();
    void copyAllElements(const Wedding_Agency& wa);
    // Methods to simplify the memory management of the object

    Wedding* findWedding(string surname) const;
    Wedding* findWedding(time_t date) const;
    // Methods for finding pointers to specific wedding based
    // on date of wedding/surname of the couple,
    // it will return nullptr if such wedding is not found

    template <typename searchMethod>
    bool isGuest(searchMethod method, string guestSurname, string guestName) const;
    template <typename searchMethod>
    bool isLocation(searchMethod method, string title) const;
    template <typename searchMethod>
    bool isSetting(searchMethod method, string title) const;
    template <typename searchMethod>
    bool isPeriod(searchMethod method, period begining, period ending) const;
    template <typename searchMethod>
    bool isNote(searchMethod method, string title) const;
    // Methods to detect if the specific elements exist in searched wedding

public:

    Wedding_Agency();
    ~Wedding_Agency();
    Wedding_Agency(const Wedding_Agency& wa);
    Wedding_Agency& operator=(const Wedding_Agency& wa);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator


    Engaged*& refClients();
    Wedding*& refWeddings();
    // Basic reference methods
    // Management of Elements

    void insertClient(string surname, time_t date, float budget);
    void insertClient(string surname, time_t date);
    // Methods for inserting a clients with given data to the list, creation of the
    // client also creates wedding in the weddings list

    template <typename searchMethod>
    void addClientInfo(searchMethod method, string groomName, string bridesName,
string contact, string email);
    // Method to add additional, optional info about a client, will not add if such
    // couple does not exists

    void removeWedding(time_t date);
    // Method to remove wedding and connected client to it from the lists
    // Method delete the data and will not remove if such elements do not exist

    template <typename searchMethod>
    void addGuest(searchMethod method, string guestSurname, string guestName,
food_preference fp);
    // Methods to add guest to specific weddings
    // Methods will not create data if such a wedding does not exist

    template <typename searchMethod>
    void removeGuest(searchMethod method, string guestSurname, string guestName);
    template <typename searchMethod>
    void removeGuest(searchMethod method, string guestSurname);
    // Methods remove guests from the specific lists, first version let us to remove
    // specific guest and second to remove everyone with the same surname
    // Methods delete the data and will not remove if such elements do not exist

    template <typename searchMethod>
    void addLocation(searchMethod method, string title, string address,
int guestCapacity, float price);
    // Methods to add Locations to specific weddings
    // Methods will not create data if such a wedding does not exist

    template <typename searchMethod>
    void removeLocation(searchMethod method, string title);
    // Methods to remove Location by their Names from specific weddings
    // Methods delete the data and will not remove if such elements do not exist

    template <typename searchMethod>
    void addWeddingSetting(searchMethod method, string title, float price, string
description);
    // Methods to add WeddingSetting to specific weddings
    // Methods will not create data if such a wedding does not exist

    template <typename searchMethod>
    void removeWeddingSetting(searchMethod method, string title);
    // Methods to remove WeddingSetting by their titles from specific weddings
    // Methods delete the data and will not remove if such elements do not exist

    template <typename searchMethod>
    void addSchedulePeriod(searchMethod method, period beginning, period ending,
string description);
    // Methods to add Schedule Period to specific weddings
    // Methods will not create data if such a wedding does not exist
    template <typename searchMethod>
    void removeSchedulePeriod(searchMethod method, period beginning, period ending);
    // Methods to remove Schedule Periods by their time spans
    // Methods delete the data and will not remove if such elements do not exist

    template <typename searchMethod>
    void addNote(searchMethod method, string title, string scribble);
    // Methods to add Note to specific weddings
    // Methods will not create data if such a wedding does not exist

    template <typename searchMethod>
    void removeNote(searchMethod method, string title);
    // Methods to remove Note by their titles
    // Methods delete the data and will not remove if such elements do not exist

    // Changing Data of Elements
    // Clients:

    template <typename searchMethod>
    void changeClientsFoodPreferences(searchMethod method, food_preference groomFP, 
food_preference bridesFP);
    template <typename searchMethod>
    void changeClientsBudget(searchMethod method, float budget);
    // Methods to change specific data of each existing clients
    // If such client does not exist nothing happens
    // Guests:

    template <typename searchMethod>
    void confirmPresence(searchMethod method, string guestSurname, string guestName);
    // Methods to change confirmation of specific guests
    // If such Guest does not exist nothing happens

    template <typename searchMethod>
    void changeFoodPreferenceGuest(searchMethod method, string guestSurname, string 
guestName, food_preference fp);
    // Methods to change food preference of specific guests
    // If such Guest does not exist nothing happens
    // Location:

    template <typename searchMethod>
    void changeReservationStateLocation(searchMethod method, string title);
    // Methods to change reservation status of specific locations
    // If such Location does not exist nothing happens
    // Wedding Setting:

    template <typename searchMethod>
    void changeReservationStateWeddingSetting(searchMethod method, string title);
    // Methods to change reservation status of specific wedding settings
    // If such wedding setting does not exist nothing happens
    // Schedule:

    template <typename searchMethod>
    void changeScheduleDescription(searchMethod method, period beginning, period 
ending, string description);
    // Methods to change description of specific schedule
    // If such schedule does not exists nothing happens
    // Note:

    template <typename searchMethod>
    void changeNote(searchMethod method, string title, string scribble);
    // Methods to change scribble of specific note
    // If such note does not exist nothing happens
```

```cpp
// Operators

// Display
// Whole list:

friend ostream& operator << (ostream& os, const Wedding_Agency& wa);
// Operator will display contents of each wedding segregated by dates
// If any list is empty program will inform about it
// Whole wedding:

template <typename searchMethod>
void displayWedding(searchMethod method);
// Method to display specific info about given wedding
// If any lists are empty program will inform about it

// Methods to check properties of object

bool ifSorted();
// Method returns true if weddings are sorted, otherwise it returns false

int countClients();
// Method returns number of clients in the list, if list is empty it returns 0

int countWeddings();
// Method returns number of weddings in the list, if list is empty it returns 0

// Main Functionality

template <typename searchMethod>
bool checkCost(searchMethod method) const;
// Methods to check if cost of the wedding is not over the budget of the client
// and display whole cost
// In situation of not defined budget method will still compare values and
// display the whole cost
// If wedding does not exist method will return false

template <typename searchMethod>
bool checkOrganisation(searchMethod method) const;
// Methods to check which organisation elements are not completed
// If list is empty program does not check them
// If wedding does not exist method will return false

template <typename searchMethod>
bool checkGuestCapacity(searchMethod method) const;
// Methods to check if reserved location aren't too small for all guests and for
// the confirmed ones
// If guests list is empty, method will not check it and return false
// If Locations list is empty, method will not check it and return false
// If wedding does not exist method will return false

template <typename searchMethod>
bool checkGuestConfirmation(searchMethod method) const;
// Methods to check how many guests have confirmed their presence
// If guests list is empty, method will not check it and return false
// If wedding does not exist method will return false
```

```
    template <typename searchMethod>
    bool checkExpensiveElements(searchMethod method) const;
    // Methods to display the most expensive location and three most expensive
    // settings
    // If there exists less than 3 settings, method will still display the possible
    // elements
    // If there does not exists both of the lists, method will return false
    // If wedding does not exist method will return false

    template <typename searchMethod>
    bool checkDishesCount(searchMethod method) const;
    // Methods to display how many dishes are needed to be prepared based on the food
    // preferences of the guests and clients
    // If guests list is empty, method will still check preferences of the clients
    // If wedding does not exist method will return false

    bool checkDateOfNearestWedding() const;
    // Method will display info of the first wedding in the list (because list of
    // weddings is sorted by dates)
    // If weddings list does not exist method will return false

    bool displayWeddingFromTimePeriod(time_t from, time_t to) const;
    // Method will display the weddings given from the specific time period
    // If weddings list does not exist method will return false
    // If there are no weddings in this time period, method will inform about it and
    // return false
};
```

2. Person

```
enum food_preference {
    meatEater,  // 0
    vegetarian, // 1
    vegan,      // 2
    diverse     // 3
};

class Person
{
protected:

    string surname;
    string name;
    food_preference foodPreference;
    // Data of each person

public:

    Person();
    Person(string surname, string name, food_preference fp);
    ~Person();
    Person(const Person& p);
    Person& operator= (const Person& p);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    string& refSurname();
    string& refName();
    food_preference& refFoodPreference();
    // Basic referential methods for access to the data of the object

    friend ostream& operator<<(ostream& os, const Person& p);
    // Method to display data of the person
};
```

a. Engaged

```
class Engaged : public Person
{
private:

    string bridesName;
    food_preference bridesFoodPreference;
    float budget;
    string contactNumber;
    string email;
    // Data of each Engaged couple

    Engaged* next;
    // Pointer to the next engaged pair in the list

    Wedding* ownWedding;
    // Pointer to own wedding from the list

public:

    Engaged(string surname);
    ~Engaged();
    Engaged(const Engaged& e);
    Engaged& operator= (const Engaged& e);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    string& refBridesName();
    food_preference& refBridesFoodPreference();
    float& refBudget();
    string& refContactNumber();
    string& refEmail();
    Wedding*& refOwnWedding();
    Engaged*& refNext();
    // Basic referential methods for access to the data of the object

    friend ostream& operator<<(ostream& os, const Engaged& p);
    // Method to display data of the couple
};
```

b. Guest

```
class Guest : public Person
{
private:

    bool confirmation;
    // Additional data of a guest describing its confirmation of the presence

    Guest* next;
    // Pointer to the next Guest in the list

public:

    Guest(string surname, string name, food_preference fp);
    ~Guest();
    Guest(const Guest& g);
    Guest& operator= (const Guest& g);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    bool& refConfirmation();
    Guest*& refNext();
    // Basic referential method for access to the data of the object

    friend ostream& operator<<(ostream& os, const Guest& g);
    // Method to display data of the guest
};
```

3. Wedding

```
class Wedding
{
private:

    time_t dateOfWedding;
    // Date of the wedding
    Engaged* groomAndBride;
    // Pointer to the Engaged couple of which is this wedding
    Guest* guestList;
    // Pointer to the guests list
    Wedding_Setting* weddingSetting;
    // Pointer to the list of wedding settings
    Location* locations;
    // Pointer to the list of the locations
    Schedule* timePeriods;
    // Pointer to the schedule of the wedding
    Note* notes;
    // Pointer to the notes
    Wedding* next;
    // Pointer to the next wedding

public:

    Wedding(time_t dateOfWedding, Engaged* groomAndBride);
    ~Wedding();
    Wedding(const Wedding& w);
    Wedding& operator= (const Wedding& w);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    time_t& refDateOfWedding();
    Engaged*& refGroomAndBride();
    Guest*& refGuestList();
    Wedding_Setting*& refWeddingSettingList();
    Location*& refLocationsList();
    Schedule*& refSchedule();
    Note*& refNotes();
    Wedding*& refNext();
    // Basic referential methods for access to the data and lists of the object

    // Special Methods
    int countGuest() const;
    // Methods counts how many guests are on the guest list
    // If its empty, methods returns 0
    int countMeatEaters() const;
    // Methods to count how many persons are meat eaters (clients and guests)
    // If guests list is empty it will count 0 for them
    int countVegetarians() const;
    // Methods to count how many persons are vegetarians (clients and guests)
    // If guests list is empty it will count 0 for them
    int countVegans() const;
    // Methods to count how many persons are vegans (clients and guests)
    // If guests list is empty it will count 0 for them
    int countDiverse() const;
    // Methods to count how many persons have diverse food preferences (clients and
    // guests)
    // If guests list is empty it will count 0 for them
```

```cpp
    int countConfirmedGuests() const;
    // Method to count how many guests confirmed their presence
    // If guest list is empty it will count 0
    int countLocations() const;
    // Method to count how many locations are on the locations list
    // If list is empty it will count 0
    float countLocationCost() const;
    // Method to count cost of every location from the list, cost of each location is
    // multiplied by the number of guests
    // If list is empty it will count 0
    int countCompletionOfLocations() const;
    // Method to count how many locations from the list still are needed to be
    // reserved
    // If list is empty it will count 0
    int countWeddingSettings() const;
    // Method to count how many wedding settings are on the wedding settings list
    // If list is empty it will count 0
    float countWeddingSettingCost() const;
    // Method to count summarised cost of every wedding setting from the list
    // If list is empty it will count 0
    int countCompletionOfWeddingSetting() const;
    // Method to count how many wedding settings from the list still are needed to be
    //prepared
    // If list is empty it will count 0
    period checkScheduleLength() const;
    // Method to check how long is the schedule
    // If list is empty it will count 0
    int countNotes() const;
    // Method to count how many notes are on the list
    // If list is empty it will count 0

    Wedding operator+(Guest& g);
    Wedding operator+(Location& l);
    Wedding operator+(Wedding_Setting& we);
    Wedding operator+(Schedule& s);
    Wedding operator+(Note& n);
    // Operators for simpler manipulation of the object when adding new elements

    friend ostream& operator<< (ostream& os, const period& p);
    friend ostream& operator<< (ostream& os, const Wedding& w);
    // Methods to display data of the wedding
};
```

4. Organisation Element

```
class Organisation_Element
{
protected:

    string title;
    bool toDo;
    // Basic data of Organisation Element

public:

    Organisation_Element();
    Organisation_Element(string title);
    ~Organisation_Element();
    Organisation_Element(const Organisation_Element& oe);
    Organisation_Element& operator= (const Organisation_Element& oe);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    string& refTitle();
    bool& refToDo();
    // Basic referential methods for access to the data and lists of the object

    friend ostream& operator<< (ostream& os, const Organisation_Element& oe);
    // Method to display data of the Organisation_Element
};
```

a. Location

```
class Location : public Organisation_Element
{
private:

    string address;
    int guestCapacity;
    float costPerGuest;
    // Additional data of Location

    Location* next;
    // Pointer to the next Location

public:

    Location(string title, string address, int guestCapacity, float costPerGuest);
    ~Location();
    Location(const Location& l);
    Location& operator= (const Location& l);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    string& refAdress();
    int& refguestCapacity();
    float& refCostPerGuest();
    Location*& refNext();
    // Basic referential methods for access to the data and lists of the object

    friend ostream& operator<< (ostream& os, const Location& l);
    // Method to display data of the Location
};
```

b. Wedding setting

```
class Wedding_Setting : public Organisation_Element
{
private:

    string description;
    float price;
    // Additional data of wedding setting

    Wedding_Setting* next;
    // Pointer to the next Location

public:

    Wedding_Setting(string title, string description, float price);
    ~Wedding_Setting();
    Wedding_Setting(const Wedding_Setting& ws);
    Wedding_Setting& operator=(const Wedding_Setting& ws);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    string& refDescription();
    float& refPrice();
    Wedding_Setting*& refNext();
    // Basic referential methods for access to the data and lists of the object

    friend ostream& operator<< (ostream& os, const Wedding_Setting& ws);
    // Method to display data of the wedding setting
};
```

5. Schedule

```
struct period {
    int hour;
    int minute;
};

class Schedule
{
private:

    period beginning;
    period ending;
    string description;
    // Basic data of the schedule

    Schedule* next;
    // Pointer to the next time span

public:

    Schedule(period beginning, period ending, string description);
    ~Schedule();
    Schedule(const Schedule& s);
    Schedule& operator=(const Schedule& s);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    period& refBeginning();
    period& refEnding();
    string& refDescription();
    Schedule*& refNext();
    // Basic referential methods for access to the data and lists of the object

    friend ostream& operator<< (ostream& os, const Schedule& s);
    // Method to display data of the time span
};

bool operator< (const period& first, const period& second);
bool operator> (const period& first, const period& second);
bool operator== (const period& first, const period& second);
//Logical methods for simplification of sorting the Schedule list
```

6. Note

```
class Note
{
private:

    string title;
    string scribble;
    // Basic data of the note

    Note* next;
    // Pointer to the next note

public:

    Note(string title, string scribble);
    ~Note();
    Note(const Note& n);
    Note& operator=(const Note& n);
    // Basic methods of a class like Constructor, Destructor, Copy Constructor and
    // Assignment operator

    string& refTitle();
    string& refScribble();
    Note*& refNext();
    // Basic referential methods for access to the data and lists of the object

    friend ostream& operator<< (ostream& os, const Note& n);
    // Method to display data of the note
};
```

4. Testing Scenarios

- `bool addingClients();`

  Scenario checks multiple situations in which we can add clients to the wedding agency in different states and with different variables.
  - If no element is added lists should be empty
  - If we add one element, method should create client object and wedding and these object should be connected (Restriction nr.3)
  - If we add more clients, wedding list should be sorted by dates
  - If we try to add client with already reserved wedding date it should not be added (Restriction nr.2)
  - If we add clients with the same surnames, they will be located on the list
  - If we add client with date before the current one, client will not be added (Restriction nr.1)

- `bool agencyIndependence();`

  Scenario checks multiple situations in which we check if wedding agency objects are independent.
  - If no element is added, agencies should be empty.
  - Adding client to the first agency will not add client to the second agency
  - Adding elements to the wedding in the second agency will not add elements in the first agency

- `bool findingWedding();`

  Scenario checks methods responsible for finding wedding with client's surname and by its date
  - When finding by date method should return proper pointer
  - When finding by surname and in list there are no more clients with this surname method should return proper pointer
  - When finding by surname and in list there are more clients with this surname, method displays every client with this surname, their info and date of their wedding, so user can still find a wedding (Restriction nr.8)

- `bool removingOfWedding();`

  Scenario checks methods responsible for removing weddings form the lists with connected to them clients.
  - If such a wedding exists, method will remove the wedding and connected to it client, delete them and will not make any more changes to the lists
  - If such a wedding does not exist, method will not make any changes

- `bool addingClientInfo();`

  Scenario checks methods responsible for adding additional info about clients
  - If such client exists, method will change the data and will not change the rest of the elements
  - If method will get negative or improper variables, method will not change anything (Restriction nr. 9)
  - If such client does not exists, method will not change anything

- `bool addingGuest();`

  Scenario checks methods responsible for adding guest to the list
    - If no guest is added, list is empty
    - If we add one guest, method should create guest and insert it properly to the list
    - If we try to add the client with the same surname and name, method will not add him to the list (Restriction nr.4)
    - If we try to add guest to not existing object, method will not do it and change anything
- `bool removingGuests();`

  Scenario checks methods responsible for removing guest from the list
    - If guest exists, method will remove it and delete it and will not change anything in the rest of the lists
    - If guests with given surnames exist, every one of them will be removed from the list and deleted and will not change the rest of the lists
    - If such a guest or group with this surname does not exist, method will not change anything
    - If such a wedding does not exist, method will not change anything

- `bool addingLocations();`

  Scenario checks methods responsible for adding locations to the list
    - If list is empty, location will be added
    - If list is already filled, location will be added
    - If we try to add location with name, already existing in the list, location will not be added (Restriction nr.5)
    - If such a wedding does not exist, method will not add anything

- `bool removingLocations();`

  Scenario checks methods responsible for removing locations from the list
    - If location exists, method will remove it and delete it and will not change anything in the rest of the lists
    - If such a location does not exist, method will not change anything
    - If such a wedding does not exist, method will not change anything

- `bool addingWeddingSettings();`

  Scenario checks methods responsible for adding wedding settings to the list
    - If list is empty, wedding setting will be added
    - If list is already filled, wedding setting will be added
    - If we try to add wedding setting with name, already existing in the list, wedding setting will not be added (Restriction nr.5)
    - If such a wedding does not exist, method will not add anything

- `bool removingWeddingSettings();`

  Scenario checks methods responsible for removing wedding settings from the list

  - If wedding setting exists, method will remove it and delete it and will not change anything in the rest of the lists
  - If such a wedding setting does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool addingSchedulePeriods();`

  Scenario checks methods responsible for adding schedule periods to the list

  - If list is empty, schedule period will be added
  - If list is already filled, schedule period will be added
  - If we try to add schedule period with time span already existing in the list, schedule period will not be added (Restriction nr.7)
  - If such a wedding does not exist, method will not add anything

- `bool removingSchedulesPeriods();`

  Scenario checks methods responsible for removing schedule periods from the list

  - If schedule period exists, method will remove it and delete it and will not change anything in the rest of the lists
  - If such a schedule period does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool addingNotes();`

  Scenario checks methods responsible for adding notes to the list

  - If list is empty, note will be added
  - If list is already filled, note will be added
  - If we try to add note with title already existing in the list, note will not be added (Restriction nr.6)
  - If such a wedding does not exist, method will not add anything

- `bool removingNotes();`

  Scenario checks methods responsible for removing notes from the list

  - If note exists, method will remove it and delete it and will not change anything in the rest of the lists
  - If such a note does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool changingClientsData();`

  Scenario checks methods responsible for changing data of clients
  - If client exists, data will be changed and rest of the lists will not be changed
  - If we try to change data for negative numbers, method will not allow it and will not change anything (Restriction nr.9)
  - If client does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool changingGuestsData();`

  Scenario checks methods responsible for changing data of guests
  - If guest exists, data will be changed and rest of the lists will not be changed
  - If we try to change data for negative numbers, method will not allow it and will not change anything (Restriction nr.9)
  - If guest does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool changingOrganisationElements();`

  Scenario checks methods responsible for changing data of organisation elements of the lists like location and wedding settings
  - If organisation element exists, data will be changed and rest of the lists will not be changed
  - If we try to change data for negative numbers, method will not allow it and will not change anything (Restriction nr.9)
  - If organisation element does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool changingSchedulesData();`

  Scenario checks methods responsible for changing data of the schedules
  - If schedule exists, data will be changed and rest of the lists will not be changed
  - If schedule does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool changingNotesData();`

  Scenario checks methods responsible for changing data of the notes
  - If note exists, data will be changed and rest of the lists will not be changed
  - If note does not exist, method will not change anything
  - If such a wedding does not exist, method will not change anything

- `bool displayTest();`

  Scenario checks display operators of every class. Display of objects should be properly sorted and easy to read. If given object does not exist, method will inform that object does not exist

- `bool costCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly count cost of the wedding, display the information and return proper true. Method will not change state of the lists
  - If lists are empty, method will return false and will not change states of any other list
  - If such a wedding does not exist, method will not check anything and return false

- `bool organisationCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly count unprepared elements and return proper value. Method will not change state of the lists
  - If lists are empty, method will return zero and will not change states of any other lists
  - If such a wedding does not exist, method will not check anything and return false

- `bool guestCapacityCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly check if amount of guests isn't too big for reserved locations
  - If lists are empty, method will return false and will not change states of any other lists
  - If any of the locations is not reserved or the list is empty, method will automatically return false
  - If such a wedding does not exist, method will not check anything and return false

- `bool guestConfirmationCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly check how many guests confirmed their presence and will display information. Method will not change any of the lists
  - If list is empty, method will return false and will not change states of any other lists
  - If such a wedding does not exist, method will not check anything and return false

- `bool expensiveElementsCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly check if which elements are the most expensive and will display them
  - If lists are empty, method will return false, will not display anything and will not change states of any other lists
  - If there is less than three organisational elements, method will still display all of them
  - If such a wedding does not exist, method will not check anything and return false

- `bool dishesCountCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly count how many guest have each of the food preferences and will display this information clearly. Method will return true
  - If guest list is empty, method will return false, will not display anything and will not change states of any other lists
  - If such a wedding does not exist, method will not check anything and return false

- `bool dateOfNearestWeddingCheckTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly display the information about the first wedding on the list and return true
  - If list is empty, method will return false and inform that there is no weddings on the list

- `bool weddingFromTimePeriodDisplayTest();`

  Scenario checks method for different situations
  - For filled lists, method will properly display weddings from given time period and return true
  - If there are no weddings in given time period, method will inform about it and return false
  - If list is empty, method will return false, will not display anything and will not change states of any other lists

- `bool negativeValuesTesting();`

  Scenario checks creation of objects and changing its data with negative numbers. None of the situation will allow for such a change. The object will not be created or data of it will not be changed (Restriction nr.9)