

TRANSPORT AND TELECOMMUNICATION INSTITUTE  
ENGINEERING FACULTY

**VET CLINIC DATABASE INTERFACE**

COURSE PROJECT  
in Programming

by

Nikita Goloviznins

student id st78436  
group 4103BDA

RIGA 2022

## TABLE OF CONTENTS

INTRODUCTION .....	3
Topic.....	3
Relevance .....	3
Implementation .....	3
Task .....	4
DESIGN .....	6
1. UI .....	6
2. Structures .....	6
3. Supporting functions .....	7
4. File manipulation functions.....	8
5. Structure output functions .....	10
6. Miscellaneous structure functions.....	10
7. Data manipulation .....	10
Filtering.....	10
Summary.....	11
Sorting.....	11
Deletion.....	11
Addition .....	11
Edit.....	11
8. Menu .....	12
TESTING .....	13
RESULTS.....	30
CONCLUSION .....	31
Upsides: .....	31
Downsides: .....	31
REFERENCES.....	32
APPENDICES .....	33
main.cpp.....	33

vet_clinic.h .....	33
vet_clinic.cpp .....	39
vet_clinic_menu.h .....	99
vet_clinic_menu.cpp .....	103

# INTRODUCTION

## Topic

Vet Clinic database interface in C++.

## Relevance

The significance of data collection, evaluation, and management has grown dramatically. In the professional world, the idea of data is always developing and shifting. It has not only provided new depths for the organizations, but it has also made operations more difficult. However, a precise data collection process, monitoring, and storage can assist organizations in dealing with these difficulties. Database management systems are critical for organisations in maintaining various databases and accessing crucial data. These management systems are used in a variety of businesses as an interface to assist users in connecting to databases. They also aid in the organisation of data so that it may be quickly accessible.

## Implementation

In Vet Clinic database there are 3 main entities: client, clients' animal and a record of client's visit. Figure 1 shows how I implemented following entities:

- Owner stands for client. This entity has 3 fields: owner\_id, full\_name, number\_of\_animals. Owner ID (and all other IDs) uses unsigned integer, which means it starts from 0, as negative numbers are usually not used in ID numeration. Full Name is an array 80 of chars, it is easier to store chars than strings in C++ and a limit of 80 characters should be enough for most of human full names. Number of Animals is an automatically generated value, it counts every accuracy of owner\_id in Animal entity, as such it cannot be a negative number and realistically it cannot reach 65535(unsigned short integer maximum) so I used unsigned short integer.
- Animal stands for client's animal. This entity has 7 fields: animal\_id, name, characteristics, gender, age, weight, owner\_id. Animal ID is unsigned integer. Name is an array 60 of char. Characteristics is animal's breed and specie, to filtering on this parameter, I made it a separate auxiliary entity. Gender is animal's gender and also a separate auxiliary entity. Age is unsigned short integer. Weight is float with restriction in methods to not make it non-positive. Owner ID is an entry to know which of the clients own this animal.
- Characteristics has 2 entities: species, breed. Species is also another auxiliary entity. Breed is an array 60 of char.

- Species has only one entity named *species* which is an array 60 of char.
- Gender has only one entity named *gender* which is an array 60 of char.
- Appointment stands for client's visit. It has 6 entities: *appointment\_id*, *owner\_id*, *animal\_id*, *date*, *reason*, *commentary*. Appointment ID, Owner ID and Animal ID are unsigned integers and relate to appropriate entities. Date is time\_t type with input and output via structure tm. Reason is a separate auxiliary entity. Commentary is an array 160 of char.
- Reason has only one entity named *reason* which is an array 80 of char.

### **Task**

My task is to create a working database interface for vet clinic. It must include binary file manipulation(saving and reading from files), an UI, data addition, removal and modification, sorting and filtering.

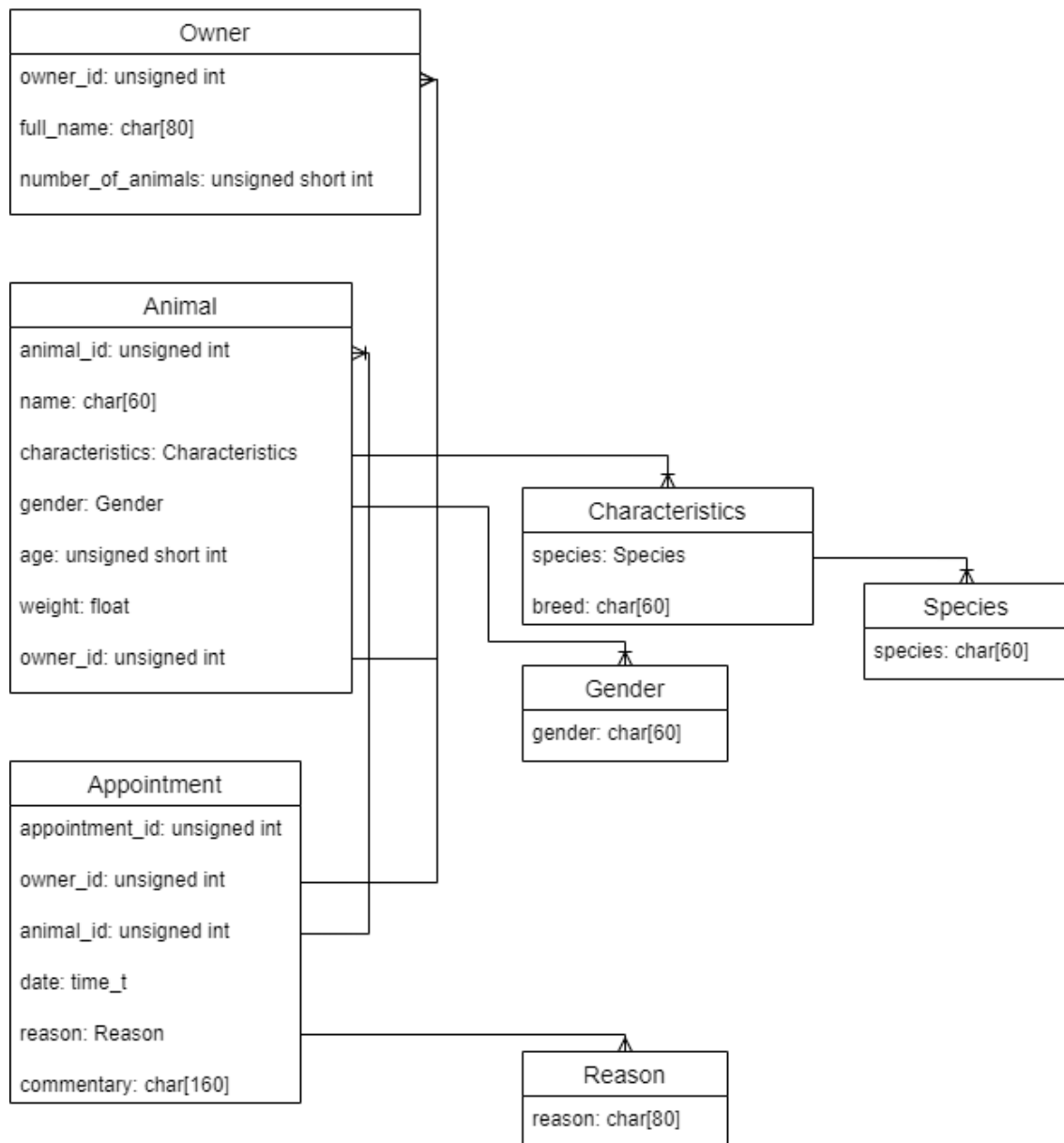
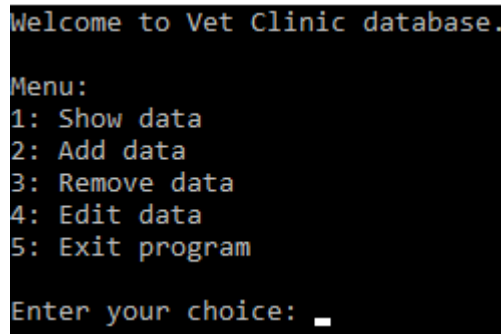


Figure 1 Entity Relation

# DESIGN

## 1. UI

My UI uses ASCII characters and navigation is done by writing a number from menu into console.

A screenshot of a terminal window showing the main menu of a program. The text is as follows:

```
Welcome to Vet Clinic database.  
  
Menu:  
1: Show data  
2: Add data  
3: Remove data  
4: Edit data  
5: Exit program  
  
Enter your choice: _
```

Figure 2 Main Menu

## 2. Structures

As described in *Implementation* section of *Introduction*, my program has 3 main and 4 auxiliary structures. From now on any code citation will use colour coding from Microsoft Visual Studio 2022 C++ syntax.

```
struct Species {  
    char species[60];  
};  
  
struct Characteristics {  
    char breed[60];  
    Species species;  
};  
  
struct Gender {  
    char gender[60];  
};  
  
struct Animal {  
    unsigned int animal_id;  
    char name[60];  
    Characteristics characteristics;  
    Gender gender;  
    unsigned short int age;  
    float weight;
```

```

    unsigned int owner_id;
};
struct Owner {
    unsigned int owner_id;
    char full_name[80];
    unsigned short int number_of_animals;
};
struct Reason {
    char reason[80];
};
struct Appointment {
    unsigned int appointment_id;
    unsigned int owner_id;
    unsigned int animal_id;
    time_t date;
    Reason reason;
    char commentary[160];
};

```

### 3. Supporting functions

I have written a few functions that directly do not use my structures but nonetheless are still very useful. I will mostly only include their declaration, for full code look into *Appendices*.

```
std::string Removespaces(std::string str);
```

This function inputs a string and returns the same string with deleted spaces on the end of string.

```

void Outputconverter(std::vector<char>& strv, int outputsized,
    unsigned int n = 1, bool formatting = false, char border = '#',
    bool endc = false);

void Outputconverter(std::string& str, int outputsized, unsigned int n = 1,
    bool formatting = false, char border = '#', bool endc = false);

```

This function inputs a string, an integer of desired size, current text line (default = 1), a boolean if you want to add a *border* at the start and at the end of each line and a boolean if it is the last line of text. This function changes your string so it can fit on the



screen with desired size and adds borders if you specify. The program works with a vector of char so it is overloaded to allow recursion.

```
void StringBouncer(std::string& str, unsigned int i);
```

This function inputs a string and crops it to specified(i) size.

```
void Clear();
```

This function clears console menu on Windows, Linux and Apple OS. This function was provided by (Joma, 2018). I have also added output of 5 line breaks in case you are running program on none of these OS.

```
void AnyKeyReturn(int& outputsize);
```

This function is used for pausing.

```
std::string bool_to_string(const bool b);
```

This function inputs a boolean and outputs string “true” or “false”.

```
time_t InputDate();
```

This function makes a user input into time\_t variable via structure tm. Format is Day.Month.Year Hour:Minute.

#### 4. File manipulation functions

All of the structure functions for saving and reading from file share a similar functionality, so I will not describe all of them. I store files in binary system by reinterpreting every data into char pointer (1 byte) and read from files using the same method: reinterpreting needed amount of byte into correct type. This method was shown to me by my programming teacher Aleksandr Sorokin (Sorokin, 2022).

```
void SaveReason(std::vector<Reason>& reason, std::string& file) {
    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(reason.empty())) {
            std::sort(reason.begin(), reason.end(), CompareReasonasc);
            for (auto& c : reason) {
                Reason* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write a reason file.\n";
    }
}
```

```

    }
}

void ReadReason(std::vector<Reason>& reason, std::string& file) {
    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Reason>().swap(reason); //makes sure vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Reason)];
            ifs.read(data, sizeof(Reason));
            Reason* pc = reinterpret_cast<Reason*>(data);
            reason.push_back(*pc);
        }
        if (!(reason.empty())) {
            std::sort(reason.begin(), reason.end(), CompareReasonasc);
        }
    }
    else {
        if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
        else {
            std::cerr << "Failed to read a reason file.\n";
        }
    }
}

```

These functions also sort vectors. Mostly by main ID but auxiliary structures are sorted by string. For that I have written custom compare functions:

```

bool CompareReasonasc(Reason& r1, Reason& r2) {
    return (strcmp(r1.reason, r2.reason) < 0);
}

```

```

void FilesInitialise(std::vector<Species>& spec,
    std::string& specfile, std::vector<Characteristics>& charac,
    std::string& characfile, std::vector<Gender>& gen, std::string& genfile,

```

```
std::vector<Animal>& animal, std::string& animalfile,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Reason>& reason, std::string& reasonfile,
std::vector<Appointment>& app, std::string& appfile);
```

This function is used on program startup for initializing all files.

## 5. Structure output functions

```
void TableOutputOwner(std::vector<Owner>& owner, int& outputsize, char& border,
std::vector<bool>& show);
```

For structure output I use these functions, they are similar for every structure. They input a vector of structure, an outputsize, character for border and a vector of boolean to decide which data to show. This functions use two other functions:

```
std::string StrHeaderOwner(int& outputsize, char& border,
std::vector<bool>& show);
```

This functions makes a header with all data names.

```
std::string StrTabOutputOwner(Owner& owner, unsigned int& i, int& outputsize,
char& border, std::vector<bool>& show);
```

This function outputs data.

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 0 # James Smith # 3 #
# *1 # 1 # Michael Smith # 1 #
# *2 # 2 # Robert Smith # 2 #
# *3 # 3 # Maria Garcia # 1 #
# *4 # 4 # David Smith # 4 #
# *5 # 5 # Maria Rodriguez # 2 #
#####
Enter any key to return_
```

Figure 3 Owner output example

## 6. Miscellaneous structure functions

```
void UpdateNoAOwner(std::string& fileowner, std::string& fileanimal);
```

This function reads owner and animal files and updates number of animals for all owners. It is called on every ReadOwner(), when a new animal has been added and every time an Owner's ID is change in either owner or animal structure.

## 7. Data manipulation

### Filtering

- Owner(= != owner\_id), Owner(= != number\_of\_animals)

- Animal(= != animal\_id), Animal(= != characteristics.species.species), Animal(= != characteristics.breed), Animal(= != gender.gender), Animal(< <= = != >= > age), Animal(< <= = != >= > weight), Animal(= != owner\_id)
- Appointment(= != appointment\_id), Appointment(= != owner\_id), Appointment(= != animal\_id), Appointment(< <= = != >= > date), Appointment(= != reason.reason)

### *Summary*

- Owner(= != number\_of\_animals)
- Animal(= != characteristics.species.species), Animal(= != characteristics.breed), Animal(= != gender.gender), Animal(< <= = != >= > age), Animal(< <= = != >= > weight)
- Appointment(= != owner\_id), Appointment(= != animal\_id), Appointment(< <= = != >= > date), Appointment(= != reason.reason)

### *Sorting*

All sorting can be applied to every value and be ascending or descending, with a priority. First select a value which should have the highest priority, then you can select it again to change from ascending to descending.

### *Deletion*

- Owner can only be deleted by its ID(owner\_id). When you delete Owner, you can also delete Animal and/or Appointment tied to its ID.
- Animal can be deleted by its and Owner's ID. When you delete animal you can also delete Appointment tied to its ID.
- Appointment can be deleted by all 3 IDs.
- All auxiliary objects are deleted by selecting an object from the list.

### *Addition*

All object addition is done by inputting correct data into the console via menu.

### *Edit*

All objects can be added via menu. Main objects are edited by ID and auxiliary are selected from the list. When you edit auxiliary objects, data will also update on other auxiliary and main objects.

## **8. Menu**

Menu is divided into 4 categories: Show, Add, Delete, Edit. With 3 subcategories in Show: show, toggle show, summary, filter, sort. All of program functionality can be accessed via these menus.

## TESTING

Test Case ID	Test Case Description	Test Steps	Expected Results	Actual Results	Pass / Fail
0	Exit program.	1. Open program 2. Enter 5 (Exit program)	Successfully exited program.	As Expected.	Pass
1	Check Show fuction for main objects with no data.	1. Open program 2. Enter 1 (Open Show Data) 3. Enter 1 (Show all main objects)	No data found for all main objects.	As Expected.	Pass
2	Check Show fuction for auxiliary objects with no data.	1. Open program 2. Enter 1 (Show data) 3. Enter 8 (Auxiliary objects) 4. Enter 1 (Species) 5. Enter (Return) 6. Enter 2 (Specie Breed) 7. Enter (Return) 8. Enter 3 (Animal Gender) 9. Enter (Return) 10. Enter 4 (Appointment Reason)	No data found for all auxiliary objects.	As Expected.	Pass
3	Check Remove function with no data.	1. Open program 2. Enter 3 (Remove data) 3. Enter 1 (Owner) 4. Enter (Return)	No data found for all objects.	As Expected.	Pass

		5. Enter 2 (Owner) 6. Enter (Return) 7. Enter 3 (Animal) 8. Enter (Return) 9. Enter 4 (Appointment) 10. Enter (Return) 11. Enter 5 (Species) 12. Enter (Return) 13. Enter 6 (Specie Breed) 14. Enter (Return) 15. Enter 7 (Appointment Reason)			
4	Check Edit function with no data.	1. Open program 2. Enter 4 (Edit data) 3. Enter 1 (Owner) 4. Enter (Return) 5. Enter 2 (Owner) 6. Enter (Return) 7. Enter 3 (Animal) 8. Enter (Return) 9. Enter 4 (Appointment) 10. Enter (Return) 11. Enter 5 (Species) 12. Enter (Return) 13. Enter 6 (Specie Breed) 14. Enter (Return)	No data found for all objects.	As Expected.	Pass

		15. Enter 7 (Appointment Reason)			
5	Check Add function for all objects.	Input: owner (owner_id = 0, full_name = Kevin Brown); species (species = Dog); characteristics (species = Dog; breed = Yorkshire Terrier); gender (gender = Female); animal (animal_id = 32; name = Lucy; characteristics = Dog, Yorkshire Terrier; gender = Female; age = 0; weight = 2.34; owner_id = 0); reason (reason = Ear infection); appointment (appointment_id = 320; owner_id = 0; animal_id = 32; date: day = 12, month = 9, year = 2022, hour = 09, minutes = 37; reason = Ear infection;	Data added, as a result Show function shows that data.	All functions added correct data except date. Shown data was the same as shown in preview in Add function (See Figure 3).	Pass



		commentary = Healing went fine) 1. Open program 2. Enter 2 (Add data) 3. Enter 1 (Owner) 4. Enter owner 5. Enter 4 (Species) 6. Enter y (yes) 7. Enter species 8. Enter y (yes) 9. Enter 5 (Specie Breed) 10. Enter 0 (Dog) 11. Enter breed 12. Enter y (yes) 13. Enter 6 (Animal Gender) 14. Enter gender 15. Enter y (yes) 16. Enter 2 (animal) 17. Enter animal 18. Enter y (yes) 19. Enter 7 (Appointment Reason) 20. Enter reason 21. Enter y (yes) 22. Enter 3 (Appointment) 23. Enter appointment 24. Enter y (yes) 25. Enter 8 (return)			
--	--	---	--	--	--

		26. Enter 1 (Show data) 27. Enter 1 (Show all main objects) 28. Enter 1 (Show objects)			
6	Check Data to be shown function for main objects.	Input: input from test case 5, saved in files. 1. Open program 2. Enter 1 (Show Data) 3. Enter 1 (Show all main objects) 4. Enter 2 (Data to be shown) 5. Enter 1 (Owner) 6. Enter 1 (Owner ID: true -> Owner ID: false) 7. Enter 3 (Number of Animals: true -> Number of Animals: false) 8. Enter 4 (Return) 9. Enter 2 (Animal) 10. Enter 1 (Animal ID: true -> Animal ID: false) 11. Enter 7 (Weight: true -> Weight: false)	Data to be shown functions correctly.	Data is correctly consealed. 1 minor visual bug at Owner data output: a missing space character (see Figure 4).	Pass

		12. Enter 8 (Owner ID: true -> Owner ID: false) 13. Enter 9 (Return) 14. Enter 3 (Appointment) 15. Enter 1 (Appointment ID: true -> Appointment ID: false) 16. Enter 2 (Owner ID: true -> Owner ID: false) 17. Enter 3 (Animal ID: true -> Animal ID: false) 18. Enter 7 (Return) 19. Enter 4 (Return) 20. Enter 1 (Show objects)			
7	Check Edit data function for auxiliary objects.	Input: Input: input from test case 5, saved in files; reason (reason = Stomach issue); gender (gender = Male); breed (breed = Munchkin); specie (specie = Cat) 1. Open program 2. Enter 4 (Edit data)	Data successfully edited and it is seen in main objects via Show function.	All data got edited, however animal.gender and appointment.reason stayed the same.	Fail

		3. Enter 7 (Appointment reason) 4. Enter 0 (Ear infection) 5. Enter reason 6. Enter 6 (Animal Gender) 7. Enter 0 (Female) 8. Enter gender 9. Enter 5 (Specie Breed) 10. Enter 2 (Breed) 11. Enter 0 (Yorkshire Terrier Dog) 12. Enter breed 13. Enter 3 (Return) 14. Enter 4 (Species) 15. Enter 0 (Dog) 16. Enter specie 17. Enter 8 (Return) 18. Enter 1 (Show data) 19. Enter 1 (Show all main objects) 20. Enter 1 (Show objects)			
8	Change name for animal, owner and change commentary for appointment.	Input: data from case 7, saved in file; full_name = John Smith; name =	Data successfully changed.	As expected.	Pass

		Spark; appointment_id = 33 1. Open program 2. Enter 4 (Edit data) 3. Enter 1 (Owner) 4. Enter 2 (Full Name) 5. Enter 0 (owner id) 6. Enter full_name 7. Enter 3 (Return) 8. Enter 2 (Animal) 9. Enter 2 (Name) 10. Enter 32 (animal id) 11. Enter name 12. Enter 9 (Return) 13. Enter 3 (Appointment) 14. Enter 1 (Appointment ID) 15. Enter 320 (appointment id) 16. Enter appointment_id 17. Enter 7 (Return) 18. Enter 8 (Return) 19. Enter 1 (Show data) 20. Enter 1 (Show all main objects) 21. Enter 1 (Show objects)			
--	--	--	--	--	--

8	Check Summary function	<p>Input: data from files (see Figure 7)</p> <ol style="list-style-type: none"> <li>1. Open program</li> <li>2. Enter 1 (Show data)</li> <li>3. Enter 1 (Show all main objects)</li> <li>4. Enter 3 (Show summary)</li> <li>5. Enter 1 (Owner)</li> <li>6. Enter 1 (Number of Animals)</li> <li>7. Enter 1 (&lt; (less than) -&gt; (&lt; (less than): Selected)</li> <li>8. Enter 7 (Value)</li> <li>9. Enter 2</li> <li>10. Enter (Return)</li> <li>11. Enter 8 (Return)</li> <li>12. Enter 2 (Return)</li> <li>13. Enter 2 (Animal)</li> <li>14. Enter 3 (Gender)</li> <li>15. Enter 1 (= (equal to) -&gt; (= (equal to): Selected)</li> <li>16. Enter 3 (Value)</li> <li>17. Enter 3 (Male)</li> <li>18. Enter (Return)</li> <li>19. Enter 4 (Return)</li> <li>20. Enter 6 (Return)</li> <li>21. Enter 3 (Appointment)</li> <li>22. Enter 3 (Date)</li> </ol>	Program shows correct summary for objects	As expected. See Figure 8.	Pass
---	------------------------	--	---	----------------------------	------

		<p>23. Enter 5 (<math>\geq</math> (greater than or equal to) <math>\rightarrow \geq</math> (greater than or equal to): Selected)</p> <p>24. Enter 7 (Value)</p> <p>25. Enter 1 (Day)</p> <p>26. Enter 2 (Month)</p> <p>27. Enter 2022 (Year)</p> <p>28. Enter 15 (Hours)</p> <p>29. Enter 12 (Minutes)</p> <p>30. Enter (Return)</p>			
9	Check Filter functions	<p>Input: data from files (see Figure 7)</p> <p>1. Open program</p> <p>2. Enter 1 (Show data)</p> <p>3. Enter 1 (Show all main objects)</p> <p>4. Enter 4 (Filters)</p> <p>5. Enter 1 (Owner)</p> <p>6. Enter 2 (Number of Animals)</p> <p>7. Enter 4 (<math>\neq</math> (not equal to) <math>\rightarrow \neq</math> (not equal to): Selected)</p> <p>8. Enter 7 (Value)</p> <p>9. Enter 1</p> <p>10. Enter 8 (Return)</p> <p>11. Enter 3 (Return)</p> <p>12. Enter 2 (Animal)</p>	Filters will correctly apply and data in Show function will be different	As expected. See Figure 9.	Pass

		13. Enter 4 (Gender) 14. Enter 1 (= (equal to) - > = (equal to): Selected) 15. Enter 3 (Value) 16. Enter 1 (Female) 17. Enter 4 (Return) 18. Enter 5 (Years of Age) 19. Enter 6 (> (greater than) - > > (greater than): Selected) 20. Enter 7 21. Enter 1 22. Enter 8 (Return) 23. Enter 8 (Return) 24. Enter 3 (Appointment) 25. Enter 5 (Reason) 26. Enter 2 (!= (not equal to) -> != (not equal to): Selected) 27. Enter 3 (Value) 28. Enter 3 (Stomach issue) 29. Enter 4 (Return) 30. Enter 6 (Return) 31. Enter 4 (Return) 32. Enter 1 (Show objects)			
10	Check Sorting function	Input: data from files (see Figure 7)	Show function will	As expected.	Pass



		1. Open program 2. Enter 1 (Show data) 3. Enter 1 (Show all main objects) 4. Enter 5 (Sorting) 5. Enter 1 (Owner) 6. Enter 3 (Number of Animals -> Number of Animals: 1 asc) 7. Enter 3 (Number of Animals: 1 asc -> Number of Animals: 1 dec) 8. Enter 2 (Full Name -> Full Name: 2 asc) 9. Enter 3 (Owner ID -> Owner ID: 3 asc) 10. Enter 3 (Owner ID: 3 asc -> Owner ID: 3 dec) 11. Enter 5 (Sort) 12. Enter 6 (Return) 13. Enter 2 (Animal) 14. Enter 6 (Years of Age -> Years of Age: 1 asc) 15. Enter 1 (Animal ID -> Animal ID: 2 asc)	correctly show data with sorting parameters.	See Figure 10. On appointment section there was a visual bug that showed incorrect data being assign a priority and sorting type, however Show function correctly showed sorted data according to input.	
--	--	--	--	--	--

		16. Enter 1 (Animal ID: 2 asc -> Animal ID: 2 dec) 17. Enter 10 (Sort) 18. Enter 11 (Return) 19. Enter 3 (Appointment) 20. Enter 4 (Date -> Date; Animal ID -> Animal ID: 1 asc) 21. Enter 4 (Date -> Date; Animal ID: 1 asc -> Animal ID: 1 dec) 22. Enter 7 (Sort) 23. Enter 8 (Return) 24. Enter 4 (Return) 25. Enter 1 (Show objects)			
--	--	--	--	--	--

Table 1 Test cases

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 0 # Kevin Brown # 1 #
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 32 # Lucy # Yorkshire Terrier # Dog # Female # 0 # 2.340000 # 0 #
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 320 # 0 # 32 # 12.8.2022 10:37 # Ear infection # Healing went fine #
#####
Enter any key to return_
```

Figure 3 Test case 5 result

```
#####
# No # Full Name #
#####
# *0 # Kevin Brown #
#####
# No # Name # Breed # Specie # Gender # Years of Age #
#####
# *0 # Lucy # Yorkshire Terrier # Dog # Female # 0 #
#####
# No # Date # Reason # Commentary #
#####
# *0 # 12.8.2022 10:37 # Ear infection # Healing went fine #
#####
Enter any key to return_
```

Figure 4 Test case 6 result

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 0 # Kevin Brown # 1 #
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 32 # Lucy # Munchkin # Cat # Female # 0 # 2.340000 # 0 #
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 320 # 0 # 32 # 12.8.2022 10:37 # Ear infection # Healing went fine #
#####
Enter any key to return
```

Figure 5 Test case 7 result

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 0 # John Smith # 1 #
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 32 # Spark # Munchkin # Cat # Female # 0 # 2.340000 # 0 #
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 33 # 0 # 32 # 12.8.2022 10:37 # Ear infection # Healing went fine #
#####
Enter any key to return_
```

Figure 6 Test case 8 result

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 0 # James Smith # 3 #
# *1 # 1 # Michael Smith # 1 #
# *2 # 2 # Robert Smith # 2 #
# *3 # 3 # Maria Garcia # 1 #
# *4 # 4 # David Smith # 4 #
# *5 # 5 # Maria Rodriguez # 2 #
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 0 # Charlie # Labrador # Dog # Male # 2 # 2.500000 # 0 #
# *1 # 1 # Bella # French Bulldog # Dog # Female # 1 # 1.500000 # 0 #
# *2 # 2 # Max # German Shepherd # Dog # Male # 3 # 3.000000 # 1 #
# *3 # 3 # Luna # None/Unknown # Dog # Female # 1 # 4.800000 # 2 #
# *4 # 4 # Oliver # Ragdoll # Cat # Male # 1 # 0.890000 # 2 #
# *5 # 5 # Lily # Maine Coon # Cat # Female # 2 # 2.523000 # 3 #
# *6 # 6 # Leo # Exotic # Cat # Male # 1 # 2.500000 # 4 #
# *7 # 7 # Lucy # None/Unknown # Cat # Female # 4 # 5.000000 # 4 #
# *8 # 8 # Shelby # Apple # Snail # Female # 0 # 2.500000 # 4 #
# *9 # 9 # Sheldon # Garden # Snail # Hermaphrodite # 1 # 2.500000 # 4 #
# *10 # 10 # Zippy # Giant African Land # Snail # Dioecious # 0 # 2.500000 # 5 #
# *11 # 11 # Lord # None/Unknown # Snail # None/Unknown # 100 # 277.542297 # 5 #
# *12 # 12 # # None/Unknown # Monkey # # 0 # 0.000000 # 0 #
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 0 # 0 # 0 # 22.5.2022 16:32 # Skin condition # Nothing out of the #
# ordinary. Healing went fine #
# *1 # 1 # 0 # 0 # 12.2.2022 12:32 # Ear infection # Bad case, almost lost an #
# ear #
# *2 # 2 # 3 # 5 # 24.3.2021 13:49 # Stomach issue # #
# *3 # 3 # 4 # 6 # 18.4.2022 0:31 # Stomach issue # Cat bit the doctor #
# *4 # 4 # 5 # 11 # 24.4.2022 0:49 # Skin condition # Never seen an animal #
# like this before #
# *5 # 5 # 1 # 2 # 1.10.2022 21:0 # None/Unknown # Gave an advice #
#####
Enter any key to return_
```

Figure 7 Data input

```
Number of records with Number of Animals < 2: 2
Enter any key to return_

Number of records with Gender = Male: 4
Enter any key to return_

Number of records with Date >= 1.1.2022 15:12 : 5
Enter any key to return_
```

Figure 8 Test case 9 result

```

#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 0 # James Smith # 3 #
# *1 # 2 # Robert Smith # 2 #
# *2 # 4 # David Smith # 4 #
# *3 # 5 # Maria Rodriguez # 2 #
#####
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 5 # Lily # Maine Coon # Cat # Female # 2 # 2.523000 # 3 #
# *1 # 7 # Lucy # None/Unknown # Cat # Female # 4 # 5.000000 # 4 #
#####
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 0 # 0 # 0 # 22.5.2022 16:32 # Skin condition # Nothing out of the #
# ordinary. Healing went fine #
# *1 # 1 # 0 # 0 # 12.2.2022 12:32 # Ear infection # Bad case, almost lost an #
# ear #
# *2 # 4 # 5 # 11 # 24.4.2022 0:49 # Skin condition # Never seen an animal #
# like this before #
# *3 # 5 # 1 # 2 # 1.10.2022 21:0 # None/Unknown # Gave an advice #
#####
Enter any key to return_

```

Figure 9 Test case 9 result

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 4 # David Smith # 4 #
# *1 # 0 # James Smith # 3 #
# *2 # 5 # Maria Rodriguez # 2 #
# *3 # 2 # Robert Smith # 2 #
# *4 # 3 # Maria Garcia # 1 #
# *5 # 1 # Michael Smith # 1 #
#####
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 12 # # None/Unknown # Monkey # # 0 # 0.000000 # 0 #
# *1 # 10 # Zippy # Giant African Land # Snail # Dioecious # 0 # 2.500000 # 5 #
# *2 # 8 # Shelby # Apple # Snail # Female # 0 # 2.500000 # 4 #
# *3 # 9 # Sheldon # Garden # Snail # Hermaphrodite # 1 # 2.500000 # 4 #
# *4 # 6 # Leo # Exotic # Cat # Male # 1 # 2.500000 # 4 #
# *5 # 4 # Oliver # Ragdoll # Cat # Male # 1 # 0.890000 # 2 #
# *6 # 3 # Luna # None/Unknown # Dog # Female # 1 # 4.800000 # 2 #
# *7 # 1 # Bella # French Bulldog # Dog # Female # 1 # 1.500000 # 0 #
# *8 # 5 # Lily # Maine Coon # Cat # Female # 2 # 2.523000 # 3 #
# *9 # 0 # Charlie # Labrador # Dog # Male # 2 # 2.500000 # 0 #
# *10 # 2 # Max # German Shepherd # Dog # Male # 3 # 3.000000 # 1 #
# *11 # 7 # Lucy # None/Unknown # Cat # Female # 4 # 5.000000 # 4 #
# *12 # 11 # Lord # None/Unknown # Snail # None/Unknown # 100 # 277.542297 # 5 #
#####
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 5 # 1 # 2 # 1.10.2022 21:0 # None/Unknown # Gave an advice #
# *1 # 0 # 0 # 0 # 22.5.2022 16:32 # Skin condition # Nothing out of the #
# ordinary. Healing went fine #
# *2 # 4 # 5 # 11 # 24.4.2022 0:49 # Skin condition # Never seen an animal #
# like this before #
# *3 # 3 # 4 # 6 # 18.4.2022 0:31 # Stomach issue # Cat bit the doctor #
# *4 # 1 # 0 # 0 # 12.2.2022 12:32 # Ear infection # Bad case, almost lost an #
# ear #
# *5 # 2 # 3 # 5 # 24.3.2021 13:49 # Stomach issue #
#####
Enter any key to return
```

Figure 10 Test case 10 result



## RESULTS

I will show how I can add a client, 2 client's animals and make a visit record.

From the main menu I select *Add data* entry, then I select *Owner*, then I write 232 for Owner ID and *James Smith* for Full Name. A conformation menu pops up, which I accept. Now I want to add 2 cats, one is Ragdoll named Boris, other is Munchkin named Player, both cats are male. For that I first need to add *Cat* in species, *Ragdoll* and *Munchkin* in breed for *Cat*, and *Male* in gender. Also I need to add a reason for appointment. After all is done this is the result:

```
#####
# No # Owner ID # Full Name # Number of Animals #
#####
# *0 # 232 # James Smith # 2 #
#####
# No # Animal ID # Name # Breed # Specie # Gender # Years of Age # Weight # #
# Owner ID #
#####
# *0 # 12 # Boris # Ragdoll # Cat # Male # 3 # 5.030000 # 232 #
# *1 # 23 # Player # Munchkin # Cat # Male # 1 # 0.900000 # 232 #
#####
# No # Appointment ID # Owner ID # Animal ID # Date # Reason # Commentary #
#####
# *0 # 1 # 232 # 12 # 15.3.2022 16:46 # Stomach issue # Nothing too serious #
#####
Enter any key to return
```

Figure 11 Program example

## CONCLUSION

In this project I have researched how to make a database in C++ and spend a lot of time coding, this is by far my biggest coding project. Maybe I got a little carried away and have written too much. I have learned many C++ tricks to write my program. In the future I could remake UI using Unicode to make it prettier.

### *Upsides:*

- I really like how I made Outputconverter() function, which enables making any text into desirable characters per line boundary.
- Toggle show is my addition as an extra function.

### *Downsides:*

- This project took a lot of time, even with preplanning I was not expecting so much work.
- I wanted to implement real entity relationship (maybe via pointers) but I chose that it would be too difficult for me.
- Menu took much longer to implement than expected. It was roughly 40% of the work.
- During testing I have found that date input is not always correct but did not have time to fix it.
- Visual bug: Sometimes table output is missing a space character, I also couldn't find what causes it as if I change function, it will consistently add an extra space character.
- Bug: When editing auxiliary object gender and reason, main objects animal and appointment do not update with edited information
- Visual bug: On appointment sorting menu output is not correct to the input, however real result is correct to the input.



## REFERENCES

- Joma, 2018. *stackoverflow*. [Online]  
Available at: <https://stackoverflow.com/questions/6486289/how-can-i-clear-console/52895729#52895729>  
[Accessed May 2022].
- Sorokin, A., 2022. *Save and read file from binary*. s.l.:s.n.

## APPENDICES

### main.cpp

```
#include "vet_clinic_menu.h"

int OutputSize = 80; //constant for output size;
char Border = '#';
std::string SpeciesFile = "species.dat";
std::string CharacteristicsFile = "characteristics.dat";
std::string GenderFile = "gender.dat";
std::string AnimalFile = "animal.dat";
std::string OwnerFile = "owner.dat";
std::string ReasonFile = "reason.dat";
std::string AppointmentFile = "appointment.dat";

int main() {
    Menu(OutputSize, Border, SpeciesFile, CharacteristicsFile, GenderFile,
        AnimalFile, OwnerFile, ReasonFile, AppointmentFile);
    return 0;
}
```

### vet\_clinic.h

```
#include <iostream>
#include <time.h>
#include <vector>
#include <string>
#include <limits>

struct Species { //auxiliary
    char species[60];
};

struct Characteristics { //auxiliary
    char breed[60];
    Species species;
};

struct Gender { //auxiliary
    char gender[60];
};

struct Animal { //main
    unsigned int animal_id;
    char name[60];
    Characteristics characteristics;
    Gender gender;
    unsigned short int age;
    float weight;
    unsigned int owner_id;
```

```

};

struct Owner { //main
    unsigned int owner_id;
    char full_name[80];
    unsigned short int number_of_animals;
};

struct Reason { //auxiliary
    char reason[80];
};

struct Appointment { //main
    unsigned int appointment_id;
    unsigned int owner_id;
    unsigned int animal_id;
    time_t date;
    Reason reason;
    char commentary[160];
};

std::string Removespaces(std::string str);

void Outputconverter(std::vector<char>& strv, int outputsize,
    unsigned int n = 1, bool formating = false, char border = '#',
    bool endc = false);

void Outputconverter(std::string& str, int outputsize, unsigned int n = 1,
    bool formating = false, char border = '#', bool endc = false);

bool CompareReasonasc(Reason& r1, Reason& r2);

void SaveReason(std::vector<Reason>& reason, std::string& file);

void ReadReason(std::vector<Reason>& reason, std::string& file);

std::string StrTabOutputReason(Reason& reason, unsigned int& i,
    int& outputsize, char& border);

std::string StrHeaderReason(int& outputsize, char& border);

void TableOutputReason(std::vector<Reason>& reason, int& outputsize,
    char border = '#');

bool CompareGenderasc(Gender& g1, Gender& g2);

void SaveGender(std::vector<Gender>& gender, std::string& file);

```

```

void ReadGender(std::vector<Gender>& gender, std::string& file);

std::string StrTabOutputGender(Gender& gender, unsigned int& i,
int& outputsize, char& border);

std::string StrHeaderGender(int& outputsize, char& border);

void TableOutputGender(std::vector<Gender>& gender, int& outputsize,
char border = '#');

bool CompareSpeciesasc(Species& s1, Species& s2);

void SaveSpecies(std::vector<Species>& species, std::string& file);

void ReadSpecies(std::vector<Species>& species, std::string& file);

std::string StrTabOutputSpecies(Species& species, unsigned int& i,
int& outputsize, char& border);

std::string StrHeaderSpecies(int& outputsize, char& border);

void TableOutputSpecies(std::vector<Species>& species, int& outputsize,
char border = '#');

bool CompareCharacteristicsbyspecasc(Characteristics& c1,
Characteristics& c2);

bool CompareCharacteristicsbybreedasc(Characteristics& c1,
Characteristics& c2);

void SaveCharacteristics(std::vector<Characteristics>& characteristics,
std::string& file);

void ReadCharacteristics(std::vector<Characteristics>& characteristics,
std::string& file);

std::string StrTabOutputCharacteristics(Characteristics& characteristics,
unsigned int& i, int& outputsize, char& border);

std::string StrHeaderCharacteristics(int& outputsize, char& border);

void TableOutputCharacteristics(std::vector<Characteristics>& characteristics,
int& outputsize, char border = '#');

void SaveAnimal(std::vector<Animal>& animal, std::string& file);

```

```

void ReadAnimal(std::vector<Animal>& animal, std::string& file);

void DeleteDataAnimal(std::vector<Animal>& animal, unsigned int id,
std::string file, unsigned short int i = 1);

bool CompareAnimalbyidasc(Animal& a1, Animal& a2);

bool CompareAnimalbyiddec(Animal& a1, Animal& a2);

bool CompareAnimalbynameasc(Animal& a1, Animal& a2);

bool CompareAnimalbynameedec(Animal& a1, Animal& a2);

bool CompareAnimalbyspeciesasc(Animal& a1, Animal& a2);

bool CompareAnimalbyspeciesdec(Animal& a1, Animal& a2);

bool CompareAnimalbybreedasc(Animal& a1, Animal& a2);

bool CompareAnimalbybreeddec(Animal& a1, Animal& a2);

bool CompareAnimalbygenderasc(Animal& a1, Animal& a2);

bool CompareAnimalbygenderdec(Animal& a1, Animal& a2);

bool CompareAnimalbyageasc(Animal& a1, Animal& a2);

bool CompareAnimalbyagedec(Animal& a1, Animal& a2);

bool CompareAnimalbyweightasc(Animal& a1, Animal& a2);

bool CompareAnimalbyweightdec(Animal& a1, Animal& a2);

bool CompareAnimalbyoidasc(Animal& a1, Animal& a2);

bool CompareAnimalbyoiddec(Animal& a1, Animal& a2);

void SortAnimal(std::vector<Animal>& result,
std::vector<unsigned short int>& prio, std::vector<std::string>& sign);

std::vector<Animal> FilterAnimal(std::vector<Animal> result,
std::vector<std::string>& sign, unsigned int& animal_id, std::string& species,
std::string& breed, std::string& gender, unsigned int& age,
float& weight, unsigned int& owner_id);

unsigned int SummaryAnimal(std::vector<Animal> result,
std::vector<std::string>& sign, std::string& species, std::string& breed,
std::string& gender, unsigned int& age, float& weight);

std::string StrTabOutputAnimal(Animal& animal, unsigned int& i,

```

```

int& outputsize, char& border, std::vector<bool>& show);

std::string StrHeaderAnimal(int& outputsize, char& border,
std::vector<bool>& show);

void TableOutputAnimal(std::vector<Animal>& animal, int& outputsize,
char& border, std::vector<bool>& show);

void SaveOwner(std::vector<Owner>& owner, std::string& file);

void ReadOwner(std::vector<Owner>& owner, std::string& file);

void DeleteDataOwner(std::vector<Owner>& owner,
std::vector<Appointment>& appointment, std::vector<Animal>& animal,
unsigned int id, std::string ofile, std::string apfile, std::string afile,
unsigned short int i = 1);

void UpdateNoAOwner(std::string& fileowner, std::string& fileanimal);

bool CompareOwnerbyidasc(Owner& o1, Owner& o2);

bool CompareOwnerbyiddec(Owner& o1, Owner& o2);

bool CompareOwnerbynameasc(Owner& o1, Owner& o2);

bool CompareOwnerbynameadec(Owner& o1, Owner& o2);

bool CompareOwnerbynoaasc(Owner& o1, Owner& o2);

bool CompareOwnerbynoadec(Owner& o1, Owner& o2);

void SortOwner(std::vector<Owner>& result,
std::vector<unsigned short int>& prio, std::vector<std::string>& sign);

std::vector<Owner> FilterOwner(std::vector<Owner> result,
std::vector<std::string>& sign, unsigned int& owner_id,
unsigned int& noa);

unsigned int SummaryOwner(std::vector<Owner> result, unsigned int& noa,
std::string& noasign);

std::string StrTabOutputOwner(Owner& owner, unsigned int& i, int& outputsize,
char& border, std::vector<bool>& show);

std::string StrHeaderOwner(int& outputsize, char& border,
std::vector<bool>& show);

void TableOutputOwner(std::vector<Owner>& owner, int& outputsize, char& border,
std::vector<bool>& show);

```

```

void SaveAppointment(std::vector<Appointment>& appointment,
std::string& file);

void ReadAppointment(std::vector<Appointment>& appointment,
std::string& file);

void DeleteDataAppointment(std::vector<Appointment>& appointment,
unsigned int id, std::string file, unsigned short int i = 1);

bool CompareAppointmentbyidasc(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyiddec(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyoidasc(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyoiddec(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyaidasc(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyaiddec(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbydateasc(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbydatedec(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyreasonasc(Appointment& ap1, Appointment& ap2);

bool CompareAppointmentbyreasondec(Appointment& ap1, Appointment& ap2);

void SortAppointment(std::vector<Appointment>& result,
std::vector<unsigned short int>& prio, std::vector<std::string>& sign);

std::vector<Appointment> FilterAppointment(std::vector<Appointment> result,
std::vector<std::string>& sign, unsigned int& appointment_id,
unsigned int& owner_id, unsigned int& animal_id, time_t& date,
std::string& reason);

unsigned int SummaryAppointment(std::vector<Appointment> result,
std::vector<std::string>& sign, unsigned int& owner_id,
unsigned int& animal_id, time_t& date, std::string& reason);

std::string StrTabOutputAppointment(Appointment& appointment, unsigned int& i,
int& outputsize, char& border, std::vector<bool>& show);

std::string StrHeaderAppointment(int& outputsize, char& border,
std::vector<bool>& show);

void TableOutputAppointment(std::vector<Appointment>& appointment,

```

```
int& outputsize, char& border, std::vector<bool>& show);
```

## **vet\_clinic.cpp**

```
#include "vet_clinic.h"
#include <iostream>
#include <vector>
#include <time.h>
#include <fstream> //for ofstream
#include <string>
#include <algorithm> //for sort

std::string Removespaces(std::string str) { //removes extra spaces at the end

    std::vector<char> strv(str.begin(), str.end());
    std::string newstr;
    if (strv.size() > 0) {
        for (auto i = strv.size() - 1; i > 1; i--)
            if (strv[i] == ' ') {
                strv.erase(strv.begin() + static_cast<__int64>(i));
            }
            else {
                break;
            }
    }
    newstr.clear();
    for (char c : strv) {
        newstr.push_back(c);
    }
    return newstr;
}

void Outputconverter(std::vector<char>& strv, int outputsize, unsigned int n,
    bool formating, char border, bool endc) { /*adds output formating and makes
    a line break if output is too long*/

    if (formating) {
        if (!(strv.size() < n * (outputsize + 1) - 1)) {
            unsigned int i = n * (outputsize + 1) - 3;
            for ( ; i > (n - 1) * (outputsize + 1); i--) {
                if (strv[i] == ' ') {
                    strv[i] = '\n';
                    strv.insert(strv.begin() + i, border);
                    if (!(i == n * (outputsize + 1) - 1)) {
                        for (unsigned int c = i; c < n * (outputsize + 1) - 2;
                            c++) {
                            strv.insert(strv.begin() + c, ' ');
                        }
                    }
                }
            }
            strv.insert(strv.begin() + (n * (outputsize + 1)), ' ');
        }
    }
}
```



```

        strv.insert(strv.begin() + (n * (outputsize + 1)), border);
        // strv.push_back(border);
        break;
    }
}
if (i == ((n - 1) * (outputsize + 1))) {
    strv.insert(strv.begin() + (n * (outputsize + 1) - 2), border);
    strv.insert(strv.begin() + (n * (outputsize + 1) - 1), '\n');
    strv.insert(strv.begin() + (n * (outputsize + 1)), ' ');
    strv.insert(strv.begin() + (n * (outputsize + 1)), border);
}
Outputconverter(strv, outputsize, n + 1, forming, border, endc);
}
else {
    if (endc) {
        strv.push_back(border);
        for (auto c = strv.size() - 1; c < n * (outputsize + 1) - 2;
             c++) {
            strv.insert(strv.begin() + static_cast<__int64>(c), ' ');
        }
        strv.push_back('\n');
    }
}
}
else {
    if (!(strv.size() < n * (outputsize + 1) - 1)) {
        unsigned int i = n * (outputsize + 1) - 1;
        for (; (i > (n - 1) * (outputsize + 1)); i--) {
            if (strv[i] == ' ') {
                strv[i] = '\n';
                if (!(i + 1 == n * (outputsize + 1))) {
                    for (unsigned int c = i; c < n * (outputsize + 1) - 1; c++) {
                        strv.insert(strv.begin() + c, ' ');
                    }
                }
                break;
            }
        }
        if (i == ((n - 1) * (outputsize + 1))) {
            strv.insert(strv.begin() + (n * (outputsize + 1) - 1), '\n');
        }
        Outputconverter(strv, outputsize, n + 1, forming, border, endc);
    }
}
}
}

void Outputconverter(std::string& str, int outputsize, unsigned int n,
    bool forming, char border, bool endc) { /*adds output forming and makes
    a line break if output is too long*/

```

```

std::vector<char> strv(str.begin(), str.end());
if (formatting) {
    if (!(strv.size() < n * (outputsize + 1) - 1)) {
        unsigned int i = n * (outputsize + 1) - 3;
        for (; i > (n - 1) * (outputsize + 1); i--) {
            if (strv[i] == ' ') {
                strv[i] = '\n';
                strv.insert(strv.begin() + i, border);
            }
            if (!(i == n * (outputsize + 1) - 1)) {
                for (unsigned int c = i; c < n * (outputsize + 1) - 2; c++) {
                    strv.insert(strv.begin() + c, ' ');
                }
            }
            strv.insert(strv.begin() + n * (outputsize + 1), ' ');
            strv.insert(strv.begin() + n * (outputsize + 1), border);
            // strv.push_back(border);
            break;
        }
    }
    if (i == ((n - 1) * (outputsize + 1))) {
        strv.insert(strv.begin() + (n * (outputsize + 1) - 2), border);
        strv.insert(strv.begin() + (n * (outputsize + 1) - 1), '\n');
        strv.insert(strv.begin() + (n * (outputsize + 1)), ' ');
        strv.insert(strv.begin() + (n * (outputsize + 1)), border);
    }
    Outputconverter(strv, outputsize, n + 1, formatting, border, endc);
}
else {
    if (endc) {
        strv.push_back(border);
        for (auto c = strv.size() - 1; c < n * (outputsize + 1) - 2; c++) {
            strv.insert(strv.begin() + static_cast<__int64>(c), ' ');
        }
        strv.push_back("\n");
    }
}
else {
    if (!(strv.size() < n * (outputsize + 1) - 1)) {
        unsigned int i = n * (outputsize + 1) - 1;
        for (; i > ((n - 1) * (outputsize + 1)); i--) {
            if (strv[i] == ' ') {
                strv[i] = '\n';
            }
            if (!(i + 1 == n * (outputsize + 1))) {
                for (unsigned int c = i; c < n * (outputsize + 1) - 1; c++) {
                    strv.insert(strv.begin() + c, ' ');
                }
            }
        }
        break;
    }
}

```

```

    }
    }
    if (i == ((n - 1) * (outputsize + 1))) {
        strv.insert(strv.begin() + (n * (outputsize + 1) - 1), '\n');
    }
    Outputconverter(strv, outputsize, n + 1, formating, border, endc);
}
}
str.clear();
for (char c : strv) {
    str.push_back(c);
}
}

bool CompareReasonasc(Reason& r1, Reason& r2) {
    return (strcmp(r1.reason, r2.reason) < 0);
}

void SaveReason(std::vector<Reason>& reason, std::string& file) {

    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(reason.empty())) {
            std::sort(reason.begin(), reason.end(), CompareReasonasc);
            for (auto& c : reason) {
                Reason* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write a reason file.\n";
    }
}

void ReadReason(std::vector<Reason>& reason, std::string& file) {

    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Reason>().swap(reason); //makes sure vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Reason)];
            ifs.read(data, sizeof(Reason));
            Reason* pc = reinterpret_cast<Reason*>(data);
            reason.push_back(*pc);
        }
        if (!(reason.empty())) {
            std::sort(reason.begin(), reason.end(), CompareReasonasc);

```

```

    }
}
else {
    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
    else {
        std::cerr << "Failed to read a reason file.\n";
    }
}
}

std::string StrTabOutputReason(Reason& reason, unsigned int& i, int& outputsize,
char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " *" + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    str = str + " " + Removespaces(reason.reason) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size()-1] == border) {
        str[str.size()-1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size()-1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border,
            true);
    }

    return str;
}

```

```

}

std::string StrHeaderReason(int& outputsize, char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;

    str = str + " Reason ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border,
            true);
    }

    return str;
}

void TableOutputReason(std::vector<Reason>& reason, int& outputsize,
    char border) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";
    std::cout << StrHeaderReason(outputsize, border);
    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";

    if (!(reason.empty())) {

```

```

        for (unsigned int n = 0; n < reason.size(); n++) {
            std::cout << StrTabOutputReason(reason[n], n, outputsize, border);
        }
    }
    else {
        std::cout << border;
        for (int n = 2; n <= outputsize - 1; n++) {
            std::cout << ' ';
        }
        std::cout << border << '\n';
    }

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
}

bool CompareGenderasc(Gender& g1, Gender& g2) {
    return (strcmp(g1.gender, g2.gender) < 0);
}

void SaveGender(std::vector<Gender>& gender, std::string& file) {

    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(gender.empty())) {
            std::sort(gender.begin(), gender.end(), CompareGenderasc);
            for (auto& c : gender) {
                Gender* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write a gender file.\n";
    }
}

void ReadGender(std::vector<Gender>& gender, std::string& file) {

    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Gender>().swap(gender); //makes sure vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Gender)];
            ifs.read(data, sizeof(Gender));
            Gender* pc = reinterpret_cast<Gender*>(data);

```

```

        gender.push_back(*pc);
    }
    if (!(gender.empty())) {
        std::sort(gender.begin(), gender.end(), CompareGenderasc);
    }
}
else {
    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
    else {
        std::cerr << "Failed to read a gender file.\n";
    }
}
}

std::string StrTabOutputGender(Gender& gender, unsigned int& i, int& outputsize,
char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " *" + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    str = str + " " + Removespaces(gender.gender) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border,
            true);
    }
}

```

```

    }

    return str;
}

std::string StrHeaderGender(int& outputsize, char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;

    str = str + " Gender ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border,
            true);
    }

    return str;
}

void TableOutputGender(std::vector<Gender>& gender, int& outputsize,
    char border) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";
    std::cout << StrHeaderGender(outputsize, border);
    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";
}

```



```

if (!(gender.empty())) {
    for (unsigned int n = 0; n < gender.size(); n++) {
        std::cout << StrTabOutputGender(gender[n], n, outputsize, border);
    }
}
else {
    std::cout << border;
    for (int n = 2; n <= outputsize - 1; n++) {
        std::cout << ' ';
    }
    std::cout << border << "\n";
}

for (int n = 1; n <= outputsize; n++) {
    std::cout << border;
}
std::cout << "\n";
}

bool CompareSpeciesasc(Species& s1, Species& s2) {
    return (strcmp(s1.species, s2.species) < 0);
}

void SaveSpecies(std::vector<Species>& species, std::string& file) {

    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(species.empty())) {
            std::sort(species.begin(), species.end(), CompareSpeciesasc);
            for (auto& c : species) {
                Species* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write a species file.\n";
    }
}

void ReadSpecies(std::vector<Species>& species, std::string& file) {

    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Species>().swap(species); //makes sure vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Species)];

```

```

        ifs.read(data, sizeof(Species));
        Species* pc = reinterpret_cast<Species*>(data);
        species.push_back(*pc);
    }
    if (!(species.empty())) {
        std::sort(species.begin(), species.end(), CompareSpeciesasc);
    }
}
else {
    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
    else {
        std::cerr << "Failed to read a species file.\n";
    }
}
}

std::string StrTabOutputSpecies(Species& species, unsigned int& i, int& outputsize,
char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " *" + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    str = str + " " + Removespaces(species.species) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size()-1] == border) {
        str[str.size()-1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size()-1] = border;
        str.push_back("\n");
    }
}

```

```

    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border,
            true);
    }

    return str;
}

std::string StrHeaderSpecies(int& outputsize, char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;

    str = str + " Specie ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back('\n');
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true, border,
            true);
    }

    return str;
}

void TableOutputSpecies(std::vector<Species>& species, int& outputsize,
    char border) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
    std::cout << StrHeaderSpecies(outputsize, border);
    for (int n = 1; n <= outputsize; n++) {

```

```

        std::cout << border;
    }
    std::cout << '\n';

    if (!(species.empty())) {
        for (unsigned int n = 0; n < species.size(); n++) {
            std::cout << StrTabOutputSpecies(species[n], n, outputsize,
            border);
        }
    }
    else {
        std::cout << border;
        for (int n = 2; n <= outputsize - 1; n++) {
            std::cout << ' ';
        }
        std::cout << border << '\n';
    }

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
}

bool CompareCharacteristicsbyspecasc(Characteristics& c1,
Characteristics& c2) {
    return (strcmp(c1.species.species, c2.species.species) < 0);
}

bool CompareCharacteristicsbybreedasc(Characteristics& c1,
Characteristics& c2) {
    return (strcmp(c1.breed, c2.breed) < 0);
}

void SaveCharacteristics(std::vector<Characteristics>& characteristics,
std::string& file) {

    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(characteristics.empty())) {
            std::sort(characteristics.begin(), characteristics.end(),
            CompareCharacteristicsbybreedasc);
            std::sort(characteristics.begin(), characteristics.end(),
            CompareCharacteristicsbyspecasc);
            for (auto& c : characteristics) {
                Characteristics* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
}

```

```

    }
    else {
        std::cerr << "Failed to write a characteristics file.\n";
    }
}

void ReadCharacteristics(std::vector<Characteristics>& characteristics,
std::string& file) {

    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Characteristics>().swap(characteristics); //makes sure
        //vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Characteristics)];
            ifs.read(data, sizeof(Characteristics));
            Characteristics* pc = reinterpret_cast<Characteristics*>(data);
            characteristics.push_back(*pc);
        }
        if (!(characteristics.empty())) {
            std::sort(characteristics.begin(), characteristics.end(),
                CompareCharacteristicsbybreedasc);
            std::sort(characteristics.begin(), characteristics.end(),
                CompareCharacteristicsbyspecasc);
        }
    }
    else {
        if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
        else {
            std::cerr << "Failed to read a characteristics file.\n";
        }
    }
}

std::string StrTabOutputCharacteristics(Characteristics& characteristics,
unsigned int& i, int& outputsize, char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " *" + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

```

```

    }

    str = str + " " + Removespaces(characteristics.breed) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    str = str + " " + Removespaces(characteristics.species.species) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size()-1] == border) {
        str[str.size()-1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size()-1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

std::string StrHeaderCharacteristics(int& outputsize, char& border) {

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;

    str = str + " Breed ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
    }

```

```

        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    str = str + " Specie ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back('\n');
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

void TableOutputCharacteristics(std::vector<Characteristics>& characteristics,
    int& outputsize, char border) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";
    std::cout << StrHeaderCharacteristics(outputsize, border);
    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";

    if (!characteristics.empty()) {
        for (unsigned int n = 0; n < characteristics.size(); n++) {
            std::cout << StrTabOutputCharacteristics(characteristics[n], n,

```

```

        outputsize, border);
    }
}
else {
    std::cout << border;
    for (int n = 2; n <= outputsize - 1; n++) {
        std::cout << ' ';
    }
    std::cout << border << "\n";
}

for (int n = 1; n <= outputsize; n++) {
    std::cout << border;
}
std::cout << "\n";
}

void SaveAnimal(std::vector<Animal>& animal, std::string& file) {

    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(animal.empty())) {
            std::sort(animal.begin(), animal.end(), CompareAnimalbyidasc);
            for (auto& c : animal) {
                Animal* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write an animal file.\n";
    }
}

void ReadAnimal(std::vector<Animal>& animal, std::string& file) {

    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Animal>().swap(animal); //makes sure vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Animal)];
            ifs.read(data, sizeof(Animal));
            Animal* pc = reinterpret_cast<Animal*>(data);
            animal.push_back(*pc);
        }
        if (!(animal.empty())) {
            std::sort(animal.begin(), animal.end(), CompareAnimalbyidasc);
        }
    }
}

```



```

    }
    else {
        if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
        else {
            std::cerr << "Failed to read an animal file.\n";
        }
    }
}

void DeleteDataAnimal(std::vector<Animal>& animal, unsigned int id,
std::string file, unsigned short int i) {
    //1 - animal_id 2 - owner_id

    size_t n = 0;
    if (i == 1) {
        while (n < animal.size()) {
            if (animal[n].animal_id == id) {
                animal.erase(animal.begin() + static_cast<__int64>(n));
                break; //there shouldn't be more than 1 entry with the same id
            }
            else {
                n++;
            }
        }
    }
    if (i == 2) {
        while (n < animal.size()) {
            if (animal[n].owner_id == id) {
                animal.erase(animal.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    SaveAnimal(animal, file);
}

bool CompareAnimalbyidasc(Animal& a1, Animal& a2) {
    return (a1.animal_id < a2.animal_id);
}

bool CompareAnimalbyiddec(Animal& a1, Animal& a2) {
    return (a1.animal_id > a2.animal_id);
}

bool CompareAnimalbynameasc(Animal& a1, Animal& a2) {
    return (strcmp(a1.name, a2.name) < 0);
}

```

```

}

bool CompareAnimalbynameasc(Animal& a1, Animal& a2) {
    return (strcmp(a1.name, a2.name) > 0);
}

bool CompareAnimalbynameascdec(Animal& a1, Animal& a2) {
    return (strcmp(a1.name, a2.name) < 0);
}

bool CompareAnimalbyspeciesasc(Animal& a1, Animal& a2) {
    return (strcmp(a1.characteristics.species.species,
        a2.characteristics.species.species) < 0);
}

bool CompareAnimalbyspeciesdec(Animal& a1, Animal& a2) {
    return (strcmp(a1.characteristics.species.species,
        a2.characteristics.species.species) > 0);
}

bool CompareAnimalbybreedasc(Animal& a1, Animal& a2) {
    return (strcmp(a1.characteristics.breed, a2.characteristics.breed) < 0);
}

bool CompareAnimalbybreeddec(Animal& a1, Animal& a2) {
    return (strcmp(a1.characteristics.breed, a2.characteristics.breed) > 0);
}

bool CompareAnimalbygenderasc(Animal& a1, Animal& a2) {
    return (strcmp(a1.gender.gender, a2.gender.gender) < 0);
}

bool CompareAnimalbygenderdec(Animal& a1, Animal& a2) {
    return (strcmp(a1.gender.gender, a2.gender.gender) > 0);
}

bool CompareAnimalbyageasc(Animal& a1, Animal& a2) {
    return (a1.age < a2.age);
}

bool CompareAnimalbyagedec(Animal& a1, Animal& a2) {
    return (a1.age > a2.age);
}

bool CompareAnimalbyweightasc(Animal& a1, Animal& a2) {
    return (a1.weight < a2.weight);
}

bool CompareAnimalbyweightdec(Animal& a1, Animal& a2) {
    return (a1.weight > a2.weight);
}

bool CompareAnimalbyoidasc(Animal& a1, Animal& a2) {
    return (a1.owner_id < a2.owner_id);
}

```

```

}

bool CompareAnimalbyiddec(Animal& a1, Animal& a2) {
    return (a1.owner_id > a2.owner_id);
}

void SortAnimal(std::vector<Animal>& result,
std::vector<unsigned short int>& prio, std::vector<std::string>& sign) {
    /*prio[/sign[] 0 - animal id 1 - name 2 - species 3 - breed 4 - gender
    5 - age 6 - weight 7 - ownerid*/
    //1 - most priority 8 - least priority

    if (!(result.empty()) or (result.size() != 1)) {
        for (unsigned short int i = 8; i >= 1; i--) {

            if (i == prio[0]) {
                if (sign[0] == "asc") {
                    std::sort(result.begin(), result.end(),
                        CompareAnimalbyidasc);
                }
                if (sign[0] == "dec") {
                    std::sort(result.begin(), result.end(),
                        CompareAnimalbyiddec);
                }
            }

            if (i == prio[1]) {
                if (sign[1] == "asc") {
                    std::sort(result.begin(), result.end(),
                        CompareAnimalbynameasc);
                }
                if (sign[1] == "dec") {
                    std::sort(result.begin(), result.end(),
                        CompareAnimalbynamedec);
                }
            }

            if (i == prio[2]) {
                if (sign[2] == "asc") {
                    std::sort(result.begin(), result.end(),
                        CompareAnimalbyspeciesasc);
                }
                if (sign[2] == "dec") {
                    std::sort(result.begin(), result.end(),
                        CompareAnimalbyspeciesdec);
                }
            }

            if (i == prio[3]) {
                if (sign[3] == "asc") {

```

```

        std::sort(result.begin(), result.end(),
            CompareAnimalbybreedasc);
    }
    if (sign[3] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbybreeddec);
    }
}

if (i == prio[4]) {
    if (sign[4] == "asc") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbygenderasc);
    }
    if (sign[4] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbygenderdec);
    }
}

if (i == prio[5]) {
    if (sign[5] == "asc") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbyageasc);
    }
    if (sign[5] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbyagedec);
    }
}

if (i == prio[6]) {
    if (sign[6] == "asc") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbyweightasc);
    }
    if (sign[6] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbyweightdec);
    }
}

if (i == prio[7]) {
    if (sign[7] == "asc") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbyoidasc);
    }
    if (sign[7] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAnimalbyoiddec);
    }
}

```

```

    }
    }
    }
    }
}

std::vector<Animal> FilterAnimal(std::vector<Animal> result,
std::vector<std::string>& sign, unsigned int& animal_id, std::string& species,
std::string& breed, std::string& gender, unsigned int& age,
float& weight, unsigned int& owner_id) {
    /*sign[] 0 - animal id 1 - species 2 - breed 3 - gender 4 - age
    5 - weight 6 - owner id*/

    size_t n;

    if (sign[0] != "") {
        n = 0;
        if (sign[0] == "=") {
            while (n < result.size()) {
                if (!(result[n].animal_id == animal_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
        if (sign[0] == "!=") {
            while (n < result.size()) {
                if (!(result[n].animal_id != animal_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
    }

    if (sign[1] != "") {
        n = 0;
        if (sign[1] == "=") {
            while (n < result.size()) {
                if (!(strcmp(result[n].characteristics.species.species,
                    species.c_str()) == 0)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
    }
}

```

```

    }
}
if (sign[1] == "!=") {
    while (n < result.size()) {
        if (!(strcmp(result[n].characteristics.species.species,
            species.c_str()) != 0)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

if (sign[2] != "") {
    n = 0;
    if (sign[2] == "=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].characteristics.breed, breed.c_str())
                == 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[2] == "!=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].characteristics.breed, breed.c_str())
                != 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

if (sign[3] != "") {
    n = 0;
    if (sign[3] == "=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].gender.gender, gender.c_str()) == 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

```

```

    }
}
}
if (sign[3] == "!=") {
    while (n < result.size()) {
        if (!(strcmp(result[n].gender.gender, gender.c_str()) != 0)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

if (sign[4] != "") {
    n = 0;
    if (sign[4] == "<") {
        while (n < result.size()) {
            if (!(result[n].age < age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == "<=") {
        while (n < result.size()) {
            if (!(result[n].age <= age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == "=") {
        while (n < result.size()) {
            if (!(result[n].age == age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == "!=") {
        while (n < result.size()) {
            if (!(result[n].age != age)) {

```

```

        result.erase(result.begin() + static_cast<__int64>(n));
    }
    else {
        n++;
    }
}
}
if (sign[4] == ">=") {
    while (n < result.size()) {
        if (!(result[n].age >= age)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[4] == ">") {
    while (n < result.size()) {
        if (!(result[n].age > age)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

if (sign[5] != "") {
    n = 0;
    if (sign[5] == "<") {
        while (n < result.size()) {
            if (!(result[n].weight < weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[5] == "<=") {
        while (n < result.size()) {
            if (!(result[n].weight <= weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

```



```

    }
    if (sign[5] == "=") {
        while (n < result.size()) {
            if (!(result[n].weight == weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[5] == "!=") {
        while (n < result.size()) {
            if (!(result[n].weight != weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[5] == ">=") {
        while (n < result.size()) {
            if (!(result[n].weight >= weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[5] == ">") {
        while (n < result.size()) {
            if (!(result[n].weight > weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

if (sign[6] != "") {
    n = 0;
    if (sign[6] == "=") {
        while (n < result.size()) {
            if (!(result[n].owner_id == owner_id)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
        }
    }
}

```

```

        else {
            n++;
        }
    }
}
if (sign[6] == "!=") {
    while (n < result.size()) {
        if (!(result[n].owner_id != owner_id)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

return result;
}

unsigned int SummaryAnimal(std::vector<Animal> result,
std::vector<std::string>& sign, std::string& species, std::string& breed,
std::string& gender, unsigned int& age, float& weight) {
    /*sign[] 0 - species 1 - breed 2 - gender 3 - age 4 - weight*/

    size_t n;

    if (sign[0] != "") {
        n = 0;
        if (sign[0] == "=") {
            while (n < result.size()) {
                if (!(strcmp(result[n].characteristics.species.species,
                    species.c_str()) == 0)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
        if (sign[0] == "!=") {
            while (n < result.size()) {
                if (!(strcmp(result[n].characteristics.species.species,
                    species.c_str()) != 0)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
    }
}

```

```

    }
}

if (sign[1] != "") {
    n = 0;
    if (sign[1] == "=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].characteristics.breed, breed.c_str())
                == 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[1] == "!=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].characteristics.breed, breed.c_str())
                != 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

if (sign[2] != "") {
    n = 0;
    if (sign[2] == "=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].gender.gender, gender.c_str()) == 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[2] == "!=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].gender.gender, gender.c_str()) != 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

```

```

    }
}

if (sign[3] != "") {
    n = 0;
    if (sign[3] == "<") {
        while (n < result.size()) {
            if (!(result[n].age < age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[3] == "<=") {
        while (n < result.size()) {
            if (!(result[n].age <= age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[3] == "=") {
        while (n < result.size()) {
            if (!(result[n].age == age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[3] == "!=") {
        while (n < result.size()) {
            if (!(result[n].age != age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[3] == ">=") {
        while (n < result.size()) {
            if (!(result[n].age >= age)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
        }
    }
}

```

```

        else {
            n++;
        }
    }
}
if (sign[3] == ">") {
    while (n < result.size()) {
        if (!(result[n].age > age)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

if (sign[4] != "") {
    n = 0;
    if (sign[4] == "<") {
        while (n < result.size()) {
            if (!(result[n].weight < weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == "<=") {
        while (n < result.size()) {
            if (!(result[n].weight <= weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == "=") {
        while (n < result.size()) {
            if (!(result[n].weight == weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == "!=") {

```

```

        while (n < result.size()) {
            if (!(result[n].weight != weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == ">=") {
        while (n < result.size()) {
            if (!(result[n].weight >= weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[4] == ">") {
        while (n < result.size()) {
            if (!(result[n].weight > weight)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

return static_cast<unsigned int>(result.size());
}

std::string StrTabOutputAnimal(Animal& animal, unsigned int& i, int& outputsize,
char& border, std::vector<bool>& show) {
    /*show[] 0 - animal id 1 - name 2 - breed 3 - species 4 - gender
    5 - age 6 - weight 7 - owner id*/

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " *" + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {

```

```

    str = str + border;
}

if (show[0]) {
    str = str + " " + std::to_string(animal.animal_id) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[1]) {
    str = str + " " + Removespaces(animal.name) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[2]) {
    str = str + " " + Removespaces(animal.characteristics.breed) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[3]) {
    str = str + " " + Removespaces(animal.characteristics.species.species)
        + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

```

```

}

if (show[4]) {
    str = str + " " + Removespaces(animal.gender.gender) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[5]) {
    str = str + " " + std::to_string(animal.age) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[6]) {
    str = str + " " + std::to_string(animal.weight) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[7]) {
    str = str + " " + std::to_string(animal.owner_id) + " ";
}
if (str[str.size()-1] == border) {
    str[str.size()-1] = ' ';
    for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
        str.push_back(' ');
    }
    str[str.size()-1] = border;
    str.push_back("\n");
}

```



```

    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

std::string StrHeaderAnimal(int& outputsize, char& border,
    std::vector<bool>& show) {
    /*show[] 0 - animal id 1 - name 2 - breed 3 - species 4 - gender
    5 - age 6 - weight 7 - owner id */

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;
    if (show[0]) {
        str = str + " Animal ID ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }

    if (show[1]) {
        str = str + " Name ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }

    if (show[2]) {
        str = str + " Breed ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {

```

```

        str = str + border;
    }
}

if (show[3]) {
    str = str + " Specie ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[4]) {
    str = str + " Gender ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[5]) {
    str = str + " Years of Age ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[6]) {
    str = str + " Weight ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

```

```

    }

    if (show[7]) {
        str = str + " Owner ID ";
    }
    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back('\n');
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

void TableOutputAnimal(std::vector<Animal>& animal, int& outputsize,
    char& border, std::vector<bool>& show) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
    std::cout << StrHeaderAnimal(outputsize, border, show);
    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';

    if (!(animal.empty())) {
        for (unsigned int n = 0; n < animal.size(); n++) {
            std::cout << StrTabOutputAnimal(animal[n], n, outputsize, border,
                show);
        }
    }
    else {
        std::cout << border;
        for (int n = 2; n <= outputsize - 1; n++) {
            std::cout << ' ';
        }
        std::cout << border << '\n';
    }

    for (int n = 1; n <= outputsize; n++) {

```

```

        std::cout << border;
    }
    std::cout << "\n";
}

void SaveOwner(std::vector<Owner>& owner, std::string& file) {
    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!(owner.empty())) {
            std::sort(owner.begin(), owner.end(), CompareOwnerbyidasc);
            for (auto& c : owner) {
                Owner* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write an owner file.\n";
    }
}

void ReadOwner(std::vector<Owner>& owner, std::string& file) {
    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Owner>().swap(owner); //makes sure vector is empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Owner)];
            ifs.read(data, sizeof(Owner));
            Owner* pc = reinterpret_cast<Owner*>(data);
            owner.push_back(*pc);
        }
        if (!(owner.empty())) {
            std::sort(owner.begin(), owner.end(), CompareOwnerbyidasc);
        }
    }
    else {
        if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
        else {
            std::cerr << "Failed to read an owner file.\n";
        }
    }
}

void DeleteDataOwner(std::vector<Owner>& owner,
    std::vector<Appointment>& appointment, std::vector<Animal>& animal,
    unsigned int id, std::string ofile, std::string apfile, std::string afile,

```

```

unsigned short int i) {
    /* i: 1 - none 2 - +delete all animals 3 - +delete all appointments
    4 - +delete all animals and appointments*/

    size_t n = 0;
    if (i == 1) {
        while (n < owner.size()) {
            if (owner[n].owner_id == id) {
                owner.erase(owner.begin() + static_cast<__int64>(n));
                break; //there shouldn't be more than 1 entry with the same id
            }
            else {
                n++;
            }
        }
    }
    if (i == 2) {
        while (n < owner.size()) {
            if (owner[n].owner_id == id) {
                owner.erase(owner.begin() + static_cast<__int64>(n));
                break;
            }
            else {
                n++;
            }
        }
        DeleteDataAnimal(animal, id, afile, 2);
    }
    if (i == 3) {
        while (n < owner.size()) {
            if (owner[n].owner_id == id) {
                owner.erase(owner.begin() + static_cast<__int64>(n));
                break;
            }
            else {
                n++;
            }
        }
        DeleteDataAppointment(appointment, id, afile, 2);
    }
    if (i == 4) {
        while (n < owner.size()) {
            if (owner[n].owner_id == id) {
                owner.erase(owner.begin() + static_cast<__int64>(n));
                break;
            }
            else {
                n++;
            }
        }
    }
}

```

```

        DeleteDataAnimal(animal, id, afile, 2);
        DeleteDataAppointment(appointment, id, apfile, 2);
    }
    SaveOwner(owner, ofile);
}

void UpdateNoAOwner(std::string& fileowner, std::string& fileanimal) {

    std::vector<Owner> owner{ };
    std::vector<Animal> animal{ };
    ReadOwner(owner, fileowner);
    ReadAnimal(animal, fileanimal);

    for (auto& o : owner) {
        o.number_of_animals = 0;
        for (auto& a : animal) {
            if (o.owner_id == a.owner_id) {
                o.number_of_animals++;
            }
        }
    }
    SaveOwner(owner, fileowner);
}

bool CompareOwnerbyidasc(Owner& o1, Owner& o2) {
    return (o1.owner_id < o2.owner_id);
}

bool CompareOwnerbyiddec(Owner& o1, Owner& o2) {
    return (o1.owner_id > o2.owner_id);
}

bool CompareOwnerbynameasc(Owner& o1, Owner& o2) {
    return (strcmp(o1.full_name, o2.full_name) < 0);
}

bool CompareOwnerbynameedec(Owner& o1, Owner& o2) {
    return (strcmp(o1.full_name, o2.full_name) > 0);
}

bool CompareOwnerbynoaasc(Owner& o1, Owner& o2) {
    return (o1.number_of_animals < o2.number_of_animals);
}

bool CompareOwnerbynoadec(Owner& o1, Owner& o2) {
    return (o1.number_of_animals > o2.number_of_animals);
}

void SortOwner(std::vector<Owner>& result,
    std::vector<unsigned short int>& prio, std::vector<std::string>& sign) {

```

```

/*prio[/sign[] 0 - owner id 1 - full name 2 - number of animals*/
//1 - most priority, 3 - least priority

if (!(result.empty()) or (result.size() != 1)) {
    for (unsigned short int i = 3; i >= 1; i--) {

        if (i == prio[0]) {
            if (sign[0] == "asc") {
                std::sort(result.begin(), result.end(),
                    CompareOwnerbyidasc);
            }
            if (sign[0] == "dec") {
                std::sort(result.begin(), result.end(),
                    CompareOwnerbyiddec);
            }
        }

        if (i == prio[1]) {
            if (sign[1] == "asc") {
                std::sort(result.begin(), result.end(),
                    CompareOwnerbynameasc);
            }
            if (sign[1] == "dec") {
                std::sort(result.begin(), result.end(),
                    CompareOwnerbynameadec);
            }
        }

        if (i == prio[2]) {
            if (sign[2] == "asc") {
                std::sort(result.begin(), result.end(),
                    CompareOwnerbynoaasc);
            }
            if (sign[2] == "dec") {
                std::sort(result.begin(), result.end(),
                    CompareOwnerbynoadec);
            }
        }
    }
}

std::vector<Owner> FilterOwner(std::vector<Owner> result,
    std::vector<std::string>& sign, unsigned int& owner_id,
    unsigned int& noa) {
    //sign[] 0 - owner id 1 - number of animals

    size_t n;

    if (sign[0] != "") {

```

```

n = 0;
if (sign[0] == "=") {
    while (n < result.size()) {
        if (!(result[n].owner_id == owner_id)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[0] == "!=") {
    while (n < result.size()) {
        if (!(result[n].owner_id != owner_id)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

if (sign[1] != "") {
    n = 0;
    if (sign[1] == "<") {
        while (n < result.size()) {
            if (!(result[n].number_of_animals < noa)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[1] == "<=") {
        while (n < result.size()) {
            if (!(result[n].number_of_animals <= noa)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[1] == "=") {
        while (n < result.size()) {
            if (!(result[n].number_of_animals == noa)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
        }
    }
}

```



```

        else {
            n++;
        }
    }
}
if (sign[1] == "!=") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals != noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[1] == ">=") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals >= noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[1] == ">") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals > noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

return result;
}

unsigned int SummaryOwner(std::vector<Owner> result, unsigned int& noa,
std::string& noassign) {

    size_t n;

    n = 0;
    if (noassign == "<") {
        while (n < result.size()) {
            if (!(result[n].number_of_animals < noa)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
        }
    }
}

```

```

    }
    else {
        n++;
    }
}
}
if (noassign == "<=") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals <= noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (noassign == "=") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals == noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (noassign == "!=") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals != noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (noassign == ">=") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals >= noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (noassign == ">") {
    while (n < result.size()) {
        if (!(result[n].number_of_animals > noa)) {
            result.erase(result.begin() + static_cast<__int64>(n));

```

```

    }
    else {
        n++;
    }
}

return static_cast<unsigned int>(result.size());
}

std::string StrTabOutputOwner(Owner& owner, unsigned int& i, int& outputsize,
char& border, std::vector<bool>& show) {
    /*show[] 0 - owner id 1 - full name 2 - number of animals*/

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " " + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (show[0]) {
        str = str + " " + std::to_string(owner.owner_id) + " ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }

    if (show[1]) {
        str = str + " " + Removespaces(owner.full_name) + " ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }
}

```

```

    }

    if (show[2]) {
        str = str + " " + std::to_string(owner.number_of_animals) + " ";
    }
    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 2; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

std::string StrHeaderOwner(int& outputsize, char& border,
std::vector<bool>& show) {
    /*show[] 0 - owner id 1 - full name 2 - number of animals*/

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;
    if (show[0]) {
        str = str + " Owner ID ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }

    if (show[1]) {
        str = str + " Full Name ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {

```

```

        str = str + border;
    }
}

if (show[2]) {
    str = str + " Number of Animals ";
}
if (str[str.size() - 1] == border) {
    str[str.size() - 1] = ' ';
    for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
        str.push_back(' ');
    }
    str[str.size() - 1] = border;
    str.push_back('\n');
}
else {
    Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
        border, true);
}

return str;
}

void TableOutputOwner(std::vector<Owner>& owner, int& outputsize, char& border,
    std::vector<bool>& show) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";
    std::cout << StrHeaderOwner(outputsize, border, show);
    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << "\n";

    if (!(owner.empty())) {
        for (unsigned int n = 0; n < owner.size(); n++) {
            std::cout << StrTabOutputOwner(owner[n], n, outputsize, border,
                show);
        }
    }
    else {
        std::cout << border;
        for (int n = 2; n <= outputsize - 1; n++) {
            std::cout << ' ';
        }
        std::cout << border << "\n";
    }
}

```

```

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
}

void SaveAppointment(std::vector<Appointment>& appointment,
std::string& file) {

    if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {
        if (!appointment.empty()) {
            std::sort(appointment.begin(), appointment.end(),
                CompareAppointmentbyidasc);
            for (auto& c : appointment) {
                Appointment* pc = &c;
                char* data = reinterpret_cast<char*>(pc);
                ofs.write(data, sizeof(c));
            }
        }
    }
    else {
        std::cerr << "Failed to write an appointment file.\n";
    }
}

void ReadAppointment(std::vector<Appointment>& appointment,
std::string& file) {

    if (std::ifstream ifs{ file, std::ios::binary | std::ios::ate }) {
        std::vector<Appointment>().swap(appointment); //makes sure vector is
        //empty
        const auto fsize = ifs.tellg();
        ifs.seekg(0);
        while (ifs.tellg() < fsize) {
            char data[sizeof(Appointment)];
            ifs.read(data, sizeof(Appointment));
            Appointment* pc = reinterpret_cast<Appointment*>(data);
            appointment.push_back(*pc);
        }
        if (!appointment.empty()) {
            std::sort(appointment.begin(), appointment.end(),
                CompareAppointmentbyidasc);
        }
    }
    else {
        if (std::ofstream ofs{ file, std::ios::binary | std::ios::trunc }) {}
        else{
            std::cerr << "Failed to read an appointment file.\n";
        }
    }
}

```

```

    }
}

void DeleteDataAppointment(std::vector<Appointment>& appointment,
    unsigned int id, std::string file, unsigned short int i) {
    /*1 - appointment_id 2 - owner_id 3 - animal_id*/

    size_t n = 0;
    if (i == 1) {
        while (n < appointment.size()) {
            if (appointment[n].appointment_id == id) {
                appointment.erase(appointment.begin() +
                    static_cast<__int64>(n));
                break; //there shouldn't be more than 1 entry with the same id
            }
            else {
                n++;
            }
        }
    }
    if (i == 2) {
        while (n < appointment.size()) {
            if (appointment[n].owner_id == id) {
                appointment.erase(appointment.begin() +
                    static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (i == 3) {
        while (n < appointment.size()) {
            if (appointment[n].animal_id == id) {
                appointment.erase(appointment.begin() +
                    static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    SaveAppointment(appointment, file);
}

bool CompareAppointmentbyidasc(Appointment& ap1, Appointment& ap2) {
    return (ap1.appointment_id < ap2.appointment_id);
}

```

```

}

bool CompareAppointmentbyiddec(Appointment& ap1, Appointment& ap2) {
    return (ap1.appointment_id > ap2.appointment_id);
}

bool CompareAppointmentbyoidasc(Appointment& ap1, Appointment& ap2) {
    return (ap1.owner_id < ap2.owner_id);
}

bool CompareAppointmentbyoiddec(Appointment& ap1, Appointment& ap2) {
    return (ap1.owner_id > ap2.owner_id);
}

bool CompareAppointmentbyaidasc(Appointment& ap1, Appointment& ap2) {
    return (ap1.animal_id < ap2.animal_id);
}

bool CompareAppointmentbyaiddec(Appointment& ap1, Appointment& ap2) {
    return (ap1.animal_id > ap2.animal_id);
}

bool CompareAppointmentbydateasc(Appointment& ap1, Appointment& ap2) {
    return (ap1.date < ap2.date);
}

bool CompareAppointmentbydatedec(Appointment& ap1, Appointment& ap2) {
    return (ap1.date > ap2.date);
}

bool CompareAppointmentbyreasonasc(Appointment& ap1, Appointment& ap2) {
    return (strcmp(ap1.reason.reason, ap2.reason.reason) < 0);
}

bool CompareAppointmentbyreasondec(Appointment& ap1, Appointment& ap2) {
    return (strcmp(ap1.reason.reason, ap2.reason.reason) > 0);
}

void SortAppointment(std::vector<Appointment>& result,
    std::vector<unsigned short int>& prio, std::vector<std::string>& sign) {
    /*prio[]/sign[] 0 - appointment id 1 - owner id 2 - animal id 3 - date
    4 - reason*/
    //1 - most priority, 5 - least priority

    if (!(result.empty()) or (result.size() != 1)) {
        for (unsigned short int i = 5; i >= 1; i--) {

            if (i == prio[0]) {
                if (sign[0] == "asc") {
                    std::sort(result.begin(), result.end(), CompareAppointmentbyidasc);
                }
            }
        }
    }
}

```



```

    }
    if (sign[0] == "dec") {
        std::sort(result.begin(), result.end(), CompareAppointmentbyiddec);
    }
}

if (i == prio[1]) {
    if (sign[1] == "asc") {
        std::sort(result.begin(), result.end(), CompareAppointmentbyoidasc);
    }
    if (sign[1] == "dec") {
        std::sort(result.begin(), result.end(), CompareAppointmentbyoiddec);
    }
}

if (i == prio[2]) {
    if (sign[2] == "asc") {
        std::sort(result.begin(), result.end(), CompareAppointmentbyaidasc);
    }
    if (sign[2] == "dec") {
        std::sort(result.begin(), result.end(), CompareAppointmentbyaiddec);
    }
}

if (i == prio[3]) {
    if (sign[3] == "asc") {
        std::sort(result.begin(), result.end(),
            CompareAppointmentbydateasc);
    }
    if (sign[3] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAppointmentbydatedec);
    }
}

if (i == prio[4]) {
    if (sign[4] == "asc") {
        std::sort(result.begin(), result.end(),
            CompareAppointmentbyreasonasc);
    }
    if (sign[4] == "dec") {
        std::sort(result.begin(), result.end(),
            CompareAppointmentbyreasondec);
    }
}
}
}

std::vector<Appointment> FilterAppointment(std::vector<Appointment> result,

```

```

std::vector<std::string>& sign, unsigned int& appointment_id,
unsigned int& owner_id, unsigned int& animal_id, time_t& date,
std::string& reason) {
    /*sign[] 0 - appointment id 1 - owner id 2 - animal id 3 - date
    4 - reason*/

    size_t n;

    if (sign[0] != "") {
        n = 0;
        if (sign[0] == "=") {
            while (n < result.size()) {
                if (!(result[n].appointment_id == appointment_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
        if (sign[0] == "!=") {
            while (n < result.size()) {
                if (!(result[n].appointment_id != appointment_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
    }

    if (sign[1] != "") {
        n = 0;
        if (sign[1] == "=") {
            while (n < result.size()) {
                if (!(result[n].owner_id == owner_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
        if (sign[1] == "!=") {
            while (n < result.size()) {
                if (!(result[n].owner_id != owner_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {

```

```

        n++;
    }
}
}

if (sign[2] != "") {
    n = 0;
    if (sign[2] == "=") {
        while (n < result.size()) {
            if (!(result[n].animal_id == animal_id)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[2] == "!=") {
        while (n < result.size()) {
            if (!(result[n].animal_id != animal_id)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

if (sign[3] != "") {
    n = 0;
    if (sign[3] == "<") {
        while (n < result.size()) {
            if (!(result[n].date < date)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[3] == "<=") {
        while (n < result.size()) {
            if (!(result[n].date <= date)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

```

```

    }
}
if (sign[3] == "=") {
    while (n < result.size()) {
        if (!(result[n].date == date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[3] == "!=") {
    while (n < result.size()) {
        if (!(result[n].date != date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[3] == ">=") {
    while (n < result.size()) {
        if (!(result[n].date >= date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
if (sign[3] == ">") {
    while (n < result.size()) {
        if (!(result[n].date > date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

if (sign[4] != "") {
    n = 0;
    if (sign[4] == "=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].reason.reason, reason.c_str()) == 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
        }
    }
}

```

```

    }
    else {
        n++;
    }
}
}
if (sign[4] == "!=") {
    while (n < result.size()) {
        if (!(strcmp(result[n].reason.reason, reason.c_str()) != 0)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

return result;
}

unsigned int SummaryAppointment(std::vector<Appointment> result,
std::vector<std::string>& sign, unsigned int& owner_id,
unsigned int& animal_id, time_t& date, std::string& reason) {
    /*sign[] 0 - owner id 1 - animal id 2 - date 3 - reason*/

    size_t n;

    if (sign[0] != "") {
        n = 0;
        if (sign[0] == "=") {
            while (n < result.size()) {
                if (!(result[n].owner_id == owner_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
        if (sign[0] == "!=") {
            while (n < result.size()) {
                if (!(result[n].owner_id != owner_id)) {
                    result.erase(result.begin() + static_cast<__int64>(n));
                }
                else {
                    n++;
                }
            }
        }
    }
}

```

```

}

if (sign[1] != "") {
    n = 0;
    if (sign[1] == "=") {
        while (n < result.size()) {
            if (!(result[n].animal_id == animal_id)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[1] == "!=") {
        while (n < result.size()) {
            if (!(result[n].animal_id != animal_id)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

if (sign[2] != "") {
    n = 0;
    if (sign[2] == "<") {
        while (n < result.size()) {
            if (!(result[n].date < date)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[2] == "<=") {
        while (n < result.size()) {
            if (!(result[n].date <= date)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
    if (sign[2] == "=") {
        while (n < result.size()) {

```

```

        if (!(result[n].date == date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}

if (sign[2] == "!=") {
    while (n < result.size()) {
        if (!(result[n].date != date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}

if (sign[2] == ">=") {
    while (n < result.size()) {
        if (!(result[n].date >= date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}

if (sign[2] == ">") {
    while (n < result.size()) {
        if (!(result[n].date > date)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}

if (sign[3] != "") {
    n = 0;
    if (sign[3] == "=") {
        while (n < result.size()) {
            if (!(strcmp(result[n].reason.reason, reason.c_str()) == 0)) {
                result.erase(result.begin() + static_cast<__int64>(n));
            }
            else {
                n++;
            }
        }
    }
}

```

```

    }
}
if (sign[3] == "!=") {
    while (n < result.size()) {
        if (!(strcmp(result[n].reason.reason, reason.c_str()) != 0)) {
            result.erase(result.begin() + static_cast<__int64>(n));
        }
        else {
            n++;
        }
    }
}
}

return static_cast<unsigned int>(result.size());
}

std::string StrTabOutputAppointment(Appointment& appointment, unsigned int& i,
int& outputsize, char& border, std::vector<bool>& show) {
    /*show[] 0 - appointment id 1 - owner id 2 - animal id 3 - date
    4 - reason 5 - commentary*/

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " *" + std::to_string(i) + " ";
    if (str.size() >= n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }

    if (show[0]) {
        str = str + " " + std::to_string(appointment.appointment_id) + " ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }

    if (show[1]) {
        str = str + " " + std::to_string(appointment.owner_id) + " ";
    }
}

```



```

    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[2]) {
    str = str + " " + std::to_string(appointment.animal_id) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[3]) {
    struct tm ptm;
    time_t temptime = appointment.date;
    localtime_s(&ptm, &temptime);
    str = str + " " + std::to_string(ptm.tm_mday) + "." +
        std::to_string(ptm.tm_mon) + "." + std::to_string(ptm.tm_year + 1900)
        + " " + std::to_string(ptm.tm_hour) + ":" + std::to_string(ptm.tm_min)
        + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[4]) {
    str = str + " " + Removespaces(appointment.reason.reason) + " ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

```

```

    }

    if (show[5]) {
        str = str + " " + Removespaces(appointment.commentary) + " ";
    }
    if (str[str.size() - 1] == border) {
        str[str.size() - 1] = ' ';
        for (auto c = str.size(); c < n * (outputsize + 1) - 2; c++) {
            str.push_back(' ');
        }
        str[str.size() - 1] = border;
        str.push_back("\n");
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

std::string StrHeaderAppointment(int& outputsize, char& border,
std::vector<bool>& show) {
    /*show[] 0 - appointment id 1 - owner id 2 - animal id 3 - date
    4 - reason 5 - commentary*/

    size_t n = 1;

    std::string str;
    str.push_back(border);
    str = str + " No " + border;
    if (show[0]) {
        str = str + " Appointment ID ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
        else {
            str = str + border;
        }
    }

    if (show[1]) {
        str = str + " Owner ID ";
        if (str.size() > n * (outputsize + 1)) {
            Outputconverter(str, outputsize, static_cast<unsigned int>(n),
                true, border);
            n = (str.size() / outputsize) + 1;
        }
    }
}

```

```

    else {
        str = str + border;
    }
}

if (show[2]) {
    str = str + " Animal ID ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[3]) {
    str = str + " Date ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[4]) {
    str = str + " Reason ";
    if (str.size() > n * (outputsize + 1)) {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n),
            true, border);
        n = (str.size() / outputsize) + 1;
    }
    else {
        str = str + border;
    }
}

if (show[5]) {
    str = str + " Commentary ";
}
if (str[str.size() - 1] == border) {
    str[str.size() - 1] = ' ';
    for (auto c = str.size(); c < n * (outputsize + 1) - 1; c++) {
        str.push_back(' ');
    }
    str[str.size() - 1] = border;
}

```

```

        str.push_back('\n');
    }
    else {
        Outputconverter(str, outputsize, static_cast<unsigned int>(n), true,
            border, true);
    }

    return str;
}

void TableOutputAppointment(std::vector<Appointment>& appointment,
    int& outputsize, char& border, std::vector<bool>& show) {

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
    std::cout << StrHeaderAppointment(outputsize, border, show);
    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';

    if (!(appointment.empty())) {
        for (unsigned int n = 0; n < appointment.size(); n++) {
            std::cout << StrTabOutputAppointment(appointment[n], n, outputsize,
                border, show);
        }
    }
    else {
        std::cout << border;
        for (int n = 2; n <= outputsize - 1; n++) {
            std::cout << ' ';
        }
        std::cout << border << '\n';
    }

    for (int n = 1; n <= outputsize; n++) {
        std::cout << border;
    }
    std::cout << '\n';
}

```

#### **vet\_clinic\_menu.h**

```

#include "vet_clinic.h"

void StringBounder(std::string& str, unsigned int i);

void Clear();

```

```

void AnyKeyReturn(int& outputsize);

time_t InputDate();

void FilesInitialise(std::vector<Species>& spec,
    std::string& specfile, std::vector<Characteristics>& charac,
    std::string& characfile, std::vector<Gender>& gen, std::string& genfile,
    std::vector<Animal>& animal, std::string& animalfile,
    std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Reason>& reason, std::string& reasonfile,
    std::vector<Appointment>& app, std::string& appfile);

std::string bool_to_string(const bool b);

void MenuDataSortingText(int& outputsize, const short int choice,
    const std::vector<unsigned short int> prio,
    const std::vector<std::string> sign);

void MenuDataSorting(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<unsigned short int> prio,
    std::vector<std::string> sign);

void MenuDataFiltersNextText(int& outputsize, const short int choice,
    const unsigned short int selected);

void MenuDataFiltersNext(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<Species>& spec,
    std::vector<Characteristics>& charac, std::vector<Gender>& gen,
    std::vector<Reason>& reason, const unsigned short int object,
    const unsigned short int data, unsigned short int selected);

void MenuDataFiltersText(int& outputsize, const short int choice);

void MenuDataFilters(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<Species>& spec,
    std::vector<Characteristics>& charac, std::vector<Gender>& gen,
    std::vector<Reason>& reason);

void MenuDataSummaryNextText(int& outputsize, const short int choice,
    const unsigned short int selected);

void MenuDataSummaryNext(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<Species>& spec,
    std::vector<Characteristics>& charac, std::vector<Gender>& gen,
    std::vector<Reason>& reason, const unsigned short int object,
    const unsigned short int data, unsigned short int selected);

```

```

void MenuDataSummaryText(int& outputsize, const short int choice);

void MenuDataSummary(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<Species>& spec,
    std::vector<Characteristics>& charac, std::vector<Gender>& gen,
    std::vector<Reason>& reason);

void MenuDataShownText(int& outputsize, const short int choice,
    std::vector<bool>& ownershow, std::vector<bool>& animalshow,
    std::vector<bool>& appshow);

void MenuDataShown(int& outputsize, const short int oldchoice,
    std::vector<bool>& ownershow, std::vector<bool>& animalshow,
    std::vector<bool>& appshow);

void MenuShowObjectsAuxiliaryText(int& outputsize);

void MenuShowObjectsAuxiliary(int& outputsize, char& border,
    std::vector<Species>& spec, std::vector<Characteristics>& charac,
    std::vector<Gender>& gen, std::vector<Reason>& reason);

void MenuShowObjectsMainText(int& outputsize, short int& choice);

void MenuShowObjectsMain(int& outputsize, char& border, short int& oldchoice,
    std::vector<Animal>& origanimal, std::vector<Owner>& origowner,
    std::vector<Appointment>& origapp, std::vector<Species>& origspec,
    std::vector<Characteristics>& origchar, std::vector<Gender>& origgen,
    std::vector<Reason>& origreason);

void MenuShowObjectsText(int& outputsize);

void MenuShowObjects(int& outputsize, char& border,
    std::vector<Species>& origspec, std::vector<Characteristics>& origchar,
    std::vector<Gender>& origgen, std::vector<Animal>& origanimal,
    std::vector<Owner>& origowner, std::vector<Reason>& origreason,
    std::vector<Appointment>& origapp);

void MenuAddObjectsText(int& outputsize);

void MenuAddOwner(int& outputsize, char& border, std::vector<Owner>& owner,
    std::string& ownerfile, std::string& animalfile);

void MenuAddAnimal(int& outputsize, char& border, std::vector<Animal>& animal,
    std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Species>& spec, std::vector<Characteristics>& charac,
    std::vector<Gender>& gen);

void MenuAddAppointment(int& outputsize, char& border,

```

```

std::vector<Appointment>& app, std::string& appfile,
std::vector<Reason>& reason);

void MenuAddSpecies(int& outputsize, std::vector<Species>& spec,
std::string& specfile);

void MenuAddCharacteristics(int& outputsize,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Species>& spec);

void MenuAddGender(int& outputsize, std::vector<Gender>& gen,
std::string& genfile);

void MenuAddReason(int& outputsize, std::vector<Reason>& reason,
std::string& reasonfile);

void MenuAddObjects(int& outputsize, char& border, std::vector<Animal>& animal,
std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile);

void MenuRemoveObjectsNextText(int& outputsize, const short int choice);

void MenuRemoveObjectsNext(int& outputsize, char& border,
const short int oldchoice, std::vector<Animal>& animal,
std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile);

void MenuRemoveObjectsText(int& outputsize);

void MenuRemoveObjects(int& outputsize, char& border,
std::vector<Animal>& animal, std::string& animalfile,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile);

void MenuEditOwner(int& outputsize, const int b, const int i,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Animal>& animal, std::string& animalfile,
std::vector<Appointment>& app, std::string& appfile);

```

```

void MenuEditAnimal(int& outputsize, const int b, const int i,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Animal>& animal, std::string& animalfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::vector<Characteristics>& charac,
std::vector<Gender>& gen);

void MenuEditAppointment(int& outputsize, const int b,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Reason>& reason);

void MenuEditCharacteristics(int& outputsize, const int b,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Species>& spec, std::vector<Animal>& animal,
std::string& animalfile);

void MenuEditObjectsNextText(int& outputsize, const short int choice);

void MenuEditObjectsNext(int& outputsize, const short int oldchoice,
std::vector<Animal>& animal, std::string& animalfile,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile);

void MenuEditObjectsText(int& outputsize);

void MenuEditObjects(int& outputsize, std::vector<Animal>& animal,
std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile);

void MenuText(int& outputsize);

void Menu(int& outputsize, char& border, std::string& specfile,
std::string& characfile, std::string& genfile, std::string& animalfile,
std::string& ownerfile, std::string& reasonfile, std::string& appfile);

```

#### **vet\_clinic\_menu.cpp**

```

#include "vet_clinic_menu.h"
#include <iostream>
#include <vector>
#include <string>
#include <algorithm> //for sort
#include <cstring>

```



```

void StringBounder(std::string& str, unsigned int i) {
    auto n = str.length();
    if (i < n) {
        n = i;
    }
    std::vector<char> strv(str.begin(), str.begin() + static_cast<__int64>(n));
    str.clear();
    for (char c : strv) {
        str.push_back(c);
    }
}

void Clear() //https://stackoverflow.com/questions/6486289/how-can-i-clear-console/52895729#52895729
{
    std::cout << "\n\n\n\n"; //if none of the below OS used
#ifdef _WIN32
    system("cls");
#elif defined (__LINUX__) || defined(__gnu_linux__) || defined(__linux__)
    system("clear");
#elif defined (__APPLE__)
    system("clear");
#endif
}

void AnyKeyReturn(int& outputsize) {
    std::string text = "Enter any key to return";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

time_t InputDate() {
    unsigned int i;
    tm date = { 0 };
    do {
        std::cout << "Day: ";
        std::cin >> i;
        if ((std::cin.fail()) or (i > 31) or (i == 0)) {
            std::cin.clear();
            std::cout << "invalid input, try again\n";
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            date.tm_mday = static_cast<int>(i);
            break;
        }
    } while (true);
}

```

```

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    std::cout << "Month: ";
    std::cin >> i;
    if ((std::cin.fail()) or (i > 12) or (i == 0)) {
        std::cin.clear();
        std::cout << "invalid input, try again\n";
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        date.tm_mon = static_cast<int>(i) - 1;
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    std::cout << "Year: ";
    std::cin >> i;
    if ((std::cin.fail()) or (i < 1970)) {           //clear input if it was bad
        std::cin.clear();
        std::cout << "invalid input, try again\n";
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        date.tm_year = static_cast<int>(i) - 1900;
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    std::cout << "Hour: ";
    std::cin >> i;
    if ((std::cin.fail()) or (i > 23)) {           //clear input if it was bad
        std::cin.clear();
        std::cout << "invalid input, try again\n";
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        date.tm_hour = static_cast<int>(i);
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    std::cout << "Minutes: ";
    std::cin >> i;

```

```

        if ((std::cin.fail()) or (i > 59)) { //clear input if it was bad
            std::cin.clear();
            std::cout << "invalid input, try again\n";
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            date.tm_min = static_cast<int>(i);
            break;
        }
    } while (true);
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    return mktime(&date);
}

void FilesInitialise(std::vector<Species>& spec,
    std::string& specfile, std::vector<Characteristics>& charac,
    std::string& characfile, std::vector<Gender>& gen, std::string& genfile,
    std::vector<Animal>& animal, std::string& animalfile,
    std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Reason>& reason, std::string& reasonfile,
    std::vector<Appointment>& app, std::string& appfile) {

    ReadSpecies(spec, specfile);
    ReadCharacteristics(charac, characfile);
    ReadGender(gen, genfile);
    ReadAnimal(animal, animalfile);
    ReadOwner(owner, ownerfile);
    ReadReason(reason, reasonfile);
    ReadAppointment(app, appfile);
}

std::string bool_to_string(const bool b) { //converts bool to true/false string
    if (b) {
        return "true";
    }
    else {
        return "false";
    }
}

void MenuDataSortingText(int& outputsize, const short int choice,
    const std::vector<unsigned short int> prio,
    const std::vector<std::string> sign) {

    std::string text;
    switch (choice) {
    case 1:
        text = "Select an object.\n\n";
        Outputconverter(text, outputsize);
    }
}

```

```

std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Owner\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Animal\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Appointment\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 2: //handling any owner case
text = "Set sorting priority and type.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Owner ID";
if (prio[0] != 0) {
    text = text + ": " + std::to_string(prio[0]) + " " + sign[0]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Full Name";
if (prio[1] != 0) {
    text = text + ": " + std::to_string(prio[1]) + " " + sign[1]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Number of Animals";
if (prio[2] != 0) {
    text = text + ": " + std::to_string(prio[2]) + " " + sign[2]
        + "\n";
}
else {

```

```

        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Reset sorting\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Sort\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

case 3: //handling any animal case
    text = "Set sorting priority and type.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Animal ID";
    if (prio[0] != 0) {
        text = text + ": " + std::to_string(prio[0]) + " " + sign[0]
            + "\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Name";
    if (prio[1] != 0) {
        text = text + ": " + std::to_string(prio[1]) + " " + sign[1]
            + "\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Specie";
    if (prio[2] != 0) {
        text = text + ": " + std::to_string(prio[2]) + " " + sign[2]
            + "\n";
    }
    else {
        text = text + '\n';
    }
}

```

```

Outputconverter(text, outputsize);
std::cout << text;
text = "4: Breed";
if (prio[3] != 0) {
    text = text + ": " + std::to_string(prio[3]) + " " + sign[3]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Gender";
if (prio[4] != 0) {
    text = text + ": " + std::to_string(prio[4]) + " " + sign[4]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Years of Age";
if (prio[5] != 0) {
    text = text + ": " + std::to_string(prio[5]) + " " + sign[5]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Weight";
if (prio[6] != 0) {
    text = text + ": " + std::to_string(prio[6]) + " " + sign[6]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Owner ID";
if (prio[7] != 0) {
    text = text + ": " + std::to_string(prio[7]) + " " + sign[7]
        + "\n";
}
else {
    text = text + '\n';
}

```

```

Outputconverter(text, outputsize);
std::cout << text;
text = "9: Reset sorting\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "10: Sort\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "11: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 4: //handling any appointment case
text = "Set sorting priority and type.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Appointment ID";
if (prio[0] != 0) {
    text = text + ": " + std::to_string(prio[0]) + " " + sign[0]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Owner ID";
if (prio[2] != 0) {
    text = text + ": " + std::to_string(prio[2]) + " " + sign[2]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Animal ID";
if (prio[3] != 0) {
    text = text + ": " + std::to_string(prio[3]) + " " + sign[3]
        + "\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Date";

```

```

        if (prio[4] != 0) {
            text = text + ": " + std::to_string(prio[4]) + " " + sign[4]
                + "\n";
        }
        else {
            text = text + '\n';
        }
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "5: Reason";
        if (prio[5] != 0) {
            text = text + ": " + std::to_string(prio[5]) + " " + sign[5]
                + "\n";
        }
        else {
            text = text + '\n';
        }
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "6: Reset sorting\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "7: Sort\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "8: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 5:
        text = "Select an object.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Owner\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Animal\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 6:
        text = "Select an object.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;

```



```

    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 7:
    text = "Select an object.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
}
}

void MenuDataSorting(int& outputsize, const short int oldchoice,
std::vector<Animal>& animal, std::vector<Owner>& owner,
std::vector<Appointment>& app, std::vector<unsigned short int> prio,
std::vector<std::string> sign) {

    short int choice;
    unsigned short int priority;
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    switch (oldchoice) {
case 1:
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
    }

```

```

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    MenuDataSorting(outputsize, 2, animal, owner, app, prio, sign);
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 2:
    Clear();
    MenuDataSorting(outputsize, 3, animal, owner, app, prio, sign);
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 3:
    Clear();
    MenuDataSorting(outputsize, 4, animal, owner, app, prio, sign);
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 4:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 4);
break;
case 2:
priority = 1;
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0){
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);

```

```

        break;
    case 2:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 3:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 4:
        std::sort(owner.begin(), owner.end(), CompareOwnerbyidasc);
        prio = std::vector<unsigned short int>(8, 0);
        sign = std::vector<std::string>(8, "dec");
        priority = 1;
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        std::cout << "\nReseted\n";
        break;
    case 5:
        SortOwner(owner, prio, sign);
        std::cout << "Done\n";
        break;
    case 6:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 6);
break;
case 3:
    priority = 1;

```

```

do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 2:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 3:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
    }
}

```

```

        break;
    case 4:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 5:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 6:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }
        else {
            sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
        }
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 7:
        if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
            prio[static_cast<unsigned __int64>(choice) - 1] = priority;
            priority++;
        }
        if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
            sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
        }

```

```

    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 8:
    if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
        prio[static_cast<unsigned __int64>(choice) - 1] = priority;
        priority++;
    }
    if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
        sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 9:
    std::sort(animal.begin(), animal.end(), CompareAnimalbyidasc);
    prio = std::vector<unsigned short int>(8, 0);
    sign = std::vector<std::string>(8, "dec");
    priority = 1;
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    std::cout << "\nReseted\n";
    break;
case 10:
    SortAnimal(animal, prio, sign);
    std::cout << "Done\n";
    break;
case 11:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 11);
break;
case 4:
    priority = 1;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

```

```

switch (choice) {
case 1:
    if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
        prio[static_cast<unsigned __int64>(choice) - 1] = priority;
        priority++;
    }
    if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
        sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 2:
    if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
        prio[static_cast<unsigned __int64>(choice) - 1] = priority;
        priority++;
    }
    if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
        sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 3:
    if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
        prio[static_cast<unsigned __int64>(choice) - 1] = priority;
        priority++;
    }
    if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
        sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 4:
    if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
        prio[static_cast<unsigned __int64>(choice) - 1] = priority;
        priority++;
    }
    if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
        sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
    }

```

```

    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 5:
    if (prio[static_cast<unsigned __int64>(choice) - 1] == 0) {
        prio[static_cast<unsigned __int64>(choice) - 1] = priority;
        priority++;
    }
    if (sign[static_cast<unsigned __int64>(choice) - 1] == "dec") {
        sign[static_cast<unsigned __int64>(choice) - 1] = "asc";
    }
    else {
        sign[static_cast<unsigned __int64>(choice) - 1] = "dec";
    }
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 6:
    std::sort(app.begin(), app.end(), CompareAppointmentbyidasc);
    prio = std::vector<unsigned short int> (8, 0);
    sign = std::vector<std::string> (8, "dec");
    priority = 1;
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    std::cout << "\nReseted\n";
    break;
case 7:
    SortAppointment(app, prio, sign);
    std::cout << "Done\n";
    break;
case 8:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 8);
break;
case 5:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {

```



```

    case 1:
        Clear();
        MenuDataSorting(outputsize, 2, animal, owner, app, prio, sign);
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 2:
        Clear();
        MenuDataSorting(outputsize, 3, animal, owner, app, prio, sign);
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 3);
break;
case 6:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataSorting(outputsize, 3, animal, owner, app, prio, sign);
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 2:
        Clear();
        MenuDataSorting(outputsize, 4, animal, owner, app, prio, sign);
        Clear();
        MenuDataSortingText(outputsize, oldchoice, prio, sign);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 7:
do {
    choice = 0;
    std::cout << "Enter your choice: ";

```

```

std::cin >> choice;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    MenuDataSorting(outputsize, 3, animal, owner, app, prio, sign);
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 2:
    Clear();
    MenuDataSorting(outputsize, 4, animal, owner, app, prio, sign);
    Clear();
    MenuDataSortingText(outputsize, oldchoice, prio, sign);
    break;
case 3:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 3);
break;
}
}

```

```

void MenuDataFiltersNextText(int& outsize, const short int choice,
const unsigned short int selected) {

```

```

    std::string text;
    switch (choice) {
case 1:
    text = "Choose sign.\n\n";
    Outputconverter(text, outsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outsize);
    std::cout << text;
    text = "1: < (less than)";
    if (selected == 1) {
        text = text + ": Selected\n";
    }
    else {
        text = text + "\n";
    }
    Outputconverter(text, outsize);
    std::cout << text;
    text = "2: <= (less than or equal to)";
    if (selected == 2) {

```

```

        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: = (equal to)";
    if (selected == 3) {
        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: != (not equal to)";
    if (selected == 4) {
        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: >= (greater than or equal to)";
    if (selected == 5) {
        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: > (greater than)";
    if (selected == 6) {
        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Value\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "8: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

```

```

case 2:
    text = "Choose sign.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: = (equal to)";
    if (selected == 3) {
        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: != (not equal to)";
    if (selected == 4) {
        text = text + ": Selected\n";
    }
    else {
        text = text + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Value\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 3: //any unsigned int case
    text = "Write a positive natural number.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 4: //any float case
    text = "Write a positive real number.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 5: //any object string case
    text = "Choose an object by writing a number after the \"*\". You can "
        " add more objects from the main menu\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 6: //any date case
    text = "Write date and time in the folowing format: Day.Month.Year "

```

```

        "Hour:Minute. Year cannot be less than 1900.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
}
}

void MenuDataFiltersNext(int& outputsize, const short int oldchoice,
std::vector<Animal>& animal, std::vector<Owner>& owner,
std::vector<Appointment>& app, std::vector<Species>& spec,
std::vector<Characteristics>& charac, std::vector<Gender>& gen,
std::vector<Reason>& reason, const unsigned short int object,
const unsigned short int data, unsigned short int selected) {
    //object: 1 - owner 2 - animal 3 - appointment
    //data: respective data to object starting from 1

    unsigned short int choice;
    std::string text;
    unsigned int intvalue1;
    unsigned int intvalue2;
    std::string strvalue1;
    std::string strvalue2;
    float floatvalue;
    time_t timevalue;
    tm date = { 0 };
    std::vector<std::string> sign(7, "");

    if (!(oldchoice == 5)) {
        MenuDataFiltersNextText(outputsize, oldchoice, 0);
    }
    switch (oldchoice) {
    case 1:
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {
            case 1:
                selected = choice;
                Clear();
                MenuDataFiltersNextText(outputsize, oldchoice, selected);
                break;
            case 2:
                selected = choice;
                Clear();
                MenuDataFiltersNextText(outputsize, oldchoice, selected);

```

```

        break;
case 3:
    selected = choice;
    Clear();
    MenuDataFiltersNextText(outputsize, oldchoice, selected);
    break;
case 4:
    selected = choice;
    Clear();
    MenuDataFiltersNextText(outputsize, oldchoice, selected);
    break;
case 5:
    selected = choice;
    Clear();
    MenuDataFiltersNextText(outputsize, oldchoice, selected);
    break;
case 6:
    selected = choice;
    Clear();
    MenuDataFiltersNextText(outputsize, oldchoice, selected);
    break;
case 7:
    if (selected != 0) {
        Clear();
        if (((object == 1) and (data == 2)) or ((object == 2) and
            (data == 5))) {
            MenuDataFiltersNext(outputsize, 3, animal, owner, app,
                spec, charac, gen, reason, object, data, selected);
        }
        if ((object == 2) and (data == 6)) {
            MenuDataFiltersNext(outputsize, 4, animal, owner, app,
                spec, charac, gen, reason, object, data, selected);
        }
        if ((object == 3) and (data == 4)) {
            MenuDataFiltersNext(outputsize, 6, animal, owner, app,
                spec, charac, gen, reason, object, data, selected);
        }
        Clear();
        MenuDataFiltersNextText(outputsize, oldchoice, selected);
    }
    else {
        text = "Choose a sign\n";
        Outputconverter(text, outputsize);
        std::cout << text;
    }
    break;
case 8:
    break;
default: std::cout << "invalid input, try again\n"; break;
}

```

```

    } while (choice != 8);
    break;
case 2:
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                selected = 3;
                Clear();
                MenuDataFiltersNextText(outputsize, oldchoice, selected);
                break;
            case 2:
                selected = 4;
                Clear();
                MenuDataFiltersNextText(outputsize, oldchoice, selected);
                break;
            case 3:
                if (selected) {
                    Clear();
                    if (((object == 2) and (data == 2)) or ((object == 2) and
                        (data == 3)) or ((object == 2) and (data == 4)) or
                        ((object == 3) and (data == 5))) {
                        MenuDataFiltersNext(outputsize, 5, animal, owner, app,
                            spec, charac, gen, reason, object, data, selected);
                    }
                    if (((object == 1) and (data == 1)) or ((object == 2) and
                        (data == 1)) or ((object == 2) and (data == 7)) or
                        ((object == 3) and (data == 1)) or ((object == 3) and
                        (data == 2)) or ((object == 3) and (data == 3))) {
                        MenuDataFiltersNext(outputsize, 3, animal, owner, app,
                            spec, charac, gen, reason, object, data, selected);
                    }
                    Clear();
                    MenuDataFiltersNextText(outputsize, oldchoice, selected);
                }
            else {
                text = "Choose a sign\n";
                Outputconverter(text, outputsize);
                std::cout << text;
            }
            break;
            case 4:
                break;
            default: std::cout << "invalid input, try again\n"; break;
        }
    } while (choice != 8);
    break;
}

```

```

    }
    } while (choice != 4);
    break;
case 3:
    do {
        std::cout << "Enter: ";
        std::cin >> intvalue1;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cout << "invalid input, try again\n";
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                           '\n');
        }
        else {
            break;
        }
    } while (true);

    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    if ((object == 1) and (data == 1)) {
        if (selected == 3) {
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {
            sign[static_cast<unsigned __int64>(data - 1)] = "!=";
        }
        owner = FilterOwner(owner, sign, intvalue1, intvalue2);
    }

    if ((object == 1) and (data == 2)) {
        if (selected == 1) {
            sign[static_cast<unsigned __int64>(data - 1)] = "<";
        }
        if (selected == 2) {
            sign[static_cast<unsigned __int64>(data - 1)] = "<=";
        }
        if (selected == 3) {
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {
            sign[static_cast<unsigned __int64>(data - 1)] = "!=";
        }
        if (selected == 5) {
            sign[static_cast<unsigned __int64>(data - 1)] = ">=";
        }
        if (selected == 6) {
            sign[static_cast<unsigned __int64>(data - 1)] = ">";
        }
        owner = FilterOwner(owner, sign, intvalue2, intvalue1);
    }

```



```

}
if ((object == 2) and (data == 1)) {
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    animal = FilterAnimal(animal, sign, intvalue1, strvalue2,
        strvalue2, strvalue2, intvalue2, floatvalue, intvalue2);
}

if ((object == 2) and (data == 5)) {
    if (selected == 1) {
        sign[static_cast<unsigned __int64>(data - 1)] = "<";
    }
    if (selected == 2) {
        sign[static_cast<unsigned __int64>(data - 1)] = "<=";
    }
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    if (selected == 5) {
        sign[static_cast<unsigned __int64>(data - 1)] = ">=";
    }
    if (selected == 6) {
        sign[static_cast<unsigned __int64>(data - 1)] = ">";
    }
    animal = FilterAnimal(animal, sign, intvalue2, strvalue2,
        strvalue2, strvalue2, intvalue1, floatvalue, intvalue2);
}

if ((object == 2) and (data == 7)) {
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    animal = FilterAnimal(animal, sign, intvalue2, strvalue2,
        strvalue2, strvalue2, intvalue2, floatvalue, intvalue1);
}

if ((object == 3) and (data == 1)) {
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {

```

```

        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    app = FilterAppointment(app, sign, intvalue1, intvalue2, intvalue2,
        timevalue, strvalue1);
}
if ((object == 3) and (data == 2)) {
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    app = FilterAppointment(app, sign, intvalue2, intvalue1, intvalue2,
        timevalue, strvalue1);
}
if ((object == 3) and (data == 3)) {
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    app = FilterAppointment(app, sign, intvalue2, intvalue2, intvalue1,
        timevalue, strvalue1);
}
break;
case 4:
do {
    std::cout << "Enter: ";
    std::cin >> floatvalue;
    if ((std::cin.fail()) or (floatvalue <= 0)) {
        std::cin.clear();
        std::cout << "invalid input, try again\n";
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
} while (!(floatvalue) or (floatvalue <= 0));

if ((object == 2) and (data == 6)) {
    if (selected == 1) {
        sign[static_cast<unsigned __int64>(data - 1)] = "<";
    }
    if (selected == 2) {
        sign[static_cast<unsigned __int64>(data - 1)] = "<=";
    }
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
}

```

```

        if (selected == 5) {
            sign[static_cast<unsigned __int64>(data - 1)] = ">=";
        }
        if (selected == 6) {
            sign[static_cast<unsigned __int64>(data - 1)] = ">";
        }
        animal = FilterAnimal(animal, sign, intvalue2, strvalue2,
                               strvalue2, strvalue2, intvalue2, floatvalue, intvalue2);
    }
    break;
case 5:
    choice = 0;
    if ((object == 2) and (data == 2)) {
        TableOutputSpecies(spec, outputsize);
        choice = static_cast<unsigned short int>(spec.size() - 1);
    }
    if ((object == 2) and (data == 3)) {
        TableOutputCharacteristics(charac, outputsize);
        choice = static_cast<unsigned short int>(charac.size() - 1);
    }
    if ((object == 2) and (data == 4)) {
        TableOutputGender(gen, outputsize);
        choice = static_cast<unsigned short int>(gen.size() - 1);
    }
    if ((object == 3) and (data == 5)) {
        TableOutputReason(reason, outputsize);
        choice = static_cast<unsigned short int>(reason.size() - 1);
    }
    MenuDataFiltersNextText(outputsize, oldchoice, 0);

    do {
        std::cout << "Enter: ";
        std::cin >> intvalue1;
        if ((std::cin.fail()) or (intvalue1 > choice)) { //clear input if it was bad
            std::cin.clear();
            std::cout << "invalid input, try again\n";
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            break;
        }
    } while (true);
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    if ((object == 2) and (data == 2)) {
        strvalue1 = spec[intvalue1].species;
        if (selected == 3) {
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {

```

```

        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    animal = FilterAnimal(animal, sign, intvalue2, strvalue1,
        strvalue2, strvalue2, intvalue2, floatvalue, intvalue2);
}
if ((object == 2) and (data == 3)) {
    strvalue1 = charac[intvalue1].breed;
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    animal = FilterAnimal(animal, sign, intvalue2, strvalue2,
        strvalue1, strvalue2, intvalue2, floatvalue, intvalue2);
}
if ((object == 2) and (data == 4)) {
    strvalue1 = gen[intvalue1].gender;
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    animal = FilterAnimal(animal, sign, intvalue2, strvalue2,
        strvalue2, strvalue1, intvalue2, floatvalue, intvalue2);
}
if ((object == 3) and (data == 5)) {
    strvalue1 = reason[intvalue1].reason;
    if (selected == 3) {
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    app = FilterAppointment(app, sign, intvalue2, intvalue2, intvalue2,
        timevalue, strvalue1);
}
break;
case 6:
    timevalue = InputDate();
    localtime_s(&date, &timevalue);
    Clear();
    if ((object == 3) and (data == 4)) {
        if (selected == 1) {
            sign[static_cast<unsigned __int64>(data - 1)] = "<";
        }
        if (selected == 2) {
            sign[static_cast<unsigned __int64>(data - 1)] = "<=";
        }
    }
}

```

```

        if (selected == 3) {
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {
            sign[static_cast<unsigned __int64>(data - 1)] = "!=";
        }
        if (selected == 5) {
            sign[static_cast<unsigned __int64>(data - 1)] = ">=";
        }
        if (selected == 6) {
            sign[static_cast<unsigned __int64>(data - 1)] = ">";
        }
        app = FilterAppointment(app, sign, intvalue2, intvalue2, intvalue2,
            timevalue, strvalue2);
    }
    break;
}
}

void MenuDataFiltersText(int& outputsize, const short int choice) {

    std::string text;
    switch (choice) {
    case 1:
        text = "Select an object.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Owner\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Animal\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Appointment\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "4: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 2: //handling any owner case
        text = "Select a data to filter.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;

```

```

text = "1: Owner ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Number of Animals\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;

case 3: //handling any animal case
text = "Select a data to filter.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Animal ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Specie\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Breed\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Gender\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Years of Age\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Weight\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Owner ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;

case 4: //handling any appointment case
text = "Select a data to filter.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;

```

```

text = "1: Appointment ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Owner ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Animal ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Date\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Reason\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 5:
text = "Select an object.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Owner\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Animal\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 6:
text = "Select an object.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Owner\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Appointment\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Return\n";

```

```

        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 7:
        text = "Select an object.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Animal\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Appointment\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    }
}

void MenuDataFilters(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<Species>& spec,
    std::vector<Characteristics>& charac, std::vector<Gender>& gen,
    std::vector<Reason>& reason) {

    short int choice;
    MenuDataFiltersText(outputsize, oldchoice);
    switch (oldchoice) {
    case 1:
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {
            case 1:
                Clear();
                MenuDataFilters(outputsize, 2, animal, owner, app, spec,
                    charac, gen, reason);
                Clear();
                MenuDataFiltersText(outputsize, oldchoice);
                break;
            case 2:

```



```

        Clear();
        MenuDataFilters(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuDataFilters(outputsize, 4, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 4:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 4);
break;
case 2:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataFiltersNext(outputsize, 2, animal, owner, app, spec,
            charac, gen, reason, 1, 1, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataFiltersNext(outputsize, 1, animal, owner, app, spec,
            charac, gen, reason, 1, 2, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 3:

```

```

do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataFiltersNext(outputsize, 2, animal, owner, app,
            spec, charac, gen, reason, 2, 1, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        if (spec.empty()) {
            std::cout << "No data for Species found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataFiltersNext(outputsize, 2, animal, owner, app,
                spec, charac, gen, reason, 2, 2, 0);
        }
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        if (charac.empty()) {
            std::cout << "No data for Breed found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataFiltersNext(outputsize, 2, animal, owner, app,
                spec, charac, gen, reason, 2, 3, 0);
        }
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        if (gen.empty()) {
            std::cout << "No data for Gender found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataFiltersNext(outputsize, 2, animal, owner, app,

```

```

        spec, charac, gen, reason, 2, 4, 0);
    }
    Clear();
    MenuDataFiltersText(outputsize, oldchoice);
    break;
case 5:
    Clear();
    MenuDataFiltersNext(outputsize, 1, animal, owner, app, spec,
        charac, gen, reason, 2, 5, 0);
    Clear();
    MenuDataFiltersText(outputsize, oldchoice);
    break;
case 6:
    Clear();
    MenuDataFiltersNext(outputsize, 1, animal, owner, app, spec,
        charac, gen, reason, 2, 6, 0);
    Clear();
    MenuDataFiltersText(outputsize, oldchoice);
    break;
case 7:
    Clear();
    MenuDataFiltersNext(outputsize, 2, animal, owner, app, spec,
        charac, gen, reason, 2, 7, 0);
    Clear();
    MenuDataFiltersText(outputsize, oldchoice);
    break;
case 8:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 8);
break;
case 4:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
case 1:
        Clear();
        MenuDataFiltersNext(outputsize, 2, animal, owner, app, spec,
            charac, gen, reason, 3, 1, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
case 2:

```

```

        Clear();
        MenuDataFiltersNext(outputsize, 2, animal, owner, app, spec,
            charac, gen, reason, 3, 2, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuDataFiltersNext(outputsize, 2, animal, owner, app, spec,
            charac, gen, reason, 3, 3, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataFiltersNext(outputsize, 1, animal, owner, app, spec,
            charac, gen, reason, 3, 4, 0);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        if (reason.empty()) {
            std::cout << "No data for Reason found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataFiltersNext(outputsize, 2, animal, owner, app,
                spec, charac, gen, reason, 3, 5, 0);
        }
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 6:
        break;
    default: std::cout << "invalid input, try again\n"; break;
} while (choice != 6);
break;
case 5:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:

```

```

        Clear();
        MenuDataFilters(outputsize, 2, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataFilters(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 6:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataFilters(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataFilters(outputsize, 4, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 7:

```

```

do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataFilters(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataFilters(outputsize, 4, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataFiltersText(outputsize, oldchoice);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
}
}

void MenuDataSummaryNextText(int& outputsize, const short int choice,
const unsigned short int selected) {

    std::string text;
    switch (choice) {
    case 1:
        text = "Choose sign.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: < (less than)";
        if (selected == 1) {
            text = text + ": Selected\n";
        }
        else {
            text = text + "\n";

```

```

    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: <= (less than or equal to)";
    if (selected == 2) {
        text = text + ": Selected\n";
    }
    else {
        text = text + "\n";
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: = (equal to)";
    if (selected == 3) {
        text = text + ": Selected\n";
    }
    else {
        text = text + "\n";
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: != (not equal to)";
    if (selected == 4) {
        text = text + ": Selected\n";
    }
    else {
        text = text + "\n";
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: >= (greater than or equal to)";
    if (selected == 5) {
        text = text + ": Selected\n";
    }
    else {
        text = text + "\n";
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: > (greater than)";
    if (selected == 6) {
        text = text + ": Selected\n";
    }
    else {
        text = text + "\n";
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Value\n";
    Outputconverter(text, outputsize);

```

```

std::cout << text;
text = "8: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 2:
text = "Choose sign.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: = (equal to)";
if (selected == 3) {
    text = text + ": Selected\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "2: != (not equal to)";
if (selected == 4) {
    text = text + ": Selected\n";
}
else {
    text = text + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Value\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 3: //any unsigned int case
text = "Write a positive natural number.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 4: //any float case
text = "Write a positive real number.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 5: //any object string case
text = "Choose an object by writing a number after the \"*\". You can "
    " add more objects from the main menu\n\n";

```



```

        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 6: //any date case
        text = "Write date and time in the folowing format: Day.Month.Year "
            "Hour:Minute. Year cannot be less than 1900.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    }
}

void MenuDataSummaryNext(int& outputsize, const short int oldchoice,
    std::vector<Animal>& animal, std::vector<Owner>& owner,
    std::vector<Appointment>& app, std::vector<Species>& spec,
    std::vector<Characteristics>& charac, std::vector<Gender>& gen,
    std::vector<Reason>& reason, const unsigned short int object,
    const unsigned short int data, unsigned short int selected) {
    //object: 1 - owner 2 - animal 3 - appointment
    //data: respective data to object starting from 1

    unsigned short int choice;
    std::string text;
    unsigned int intvalue1;
    unsigned int intvalue2;
    std::string strvalue1;
    std::string strvalue2;
    float floatvalue;
    time_t timevalue;
    tm date = { 0 };
    std::vector<std::string> sign(5, "");

    if (!(oldchoice == 5)) {
        MenuDataSummaryNextText(outputsize, oldchoice, 0);
    }
    switch (oldchoice) {
    case 1:
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {
            case 1:
                selected = choice;
                Clear();
                MenuDataSummaryNextText(outputsize, oldchoice, selected);

```

```

        break;
case 2:
    selected = choice;
    Clear();
    MenuDataSummaryNextText(outputsize, oldchoice, selected);
    break;
case 3:
    selected = choice;
    Clear();
    MenuDataSummaryNextText(outputsize, oldchoice, selected);
    break;
case 4:
    selected = choice;
    Clear();
    MenuDataSummaryNextText(outputsize, oldchoice, selected);
    break;
case 5:
    selected = choice;
    Clear();
    MenuDataSummaryNextText(outputsize, oldchoice, selected);
    break;
case 6:
    selected = choice;
    Clear();
    MenuDataSummaryNextText(outputsize, oldchoice, selected);
    break;
case 7:
    if (selected != 0) {
        Clear();
        if (((object == 1) and (data == 1)) or ((object == 2) and
            (data == 4))) {
            MenuDataSummaryNext(outputsize, 3, animal, owner, app,
                spec, charac, gen, reason, object, data, selected);
        }
        if ((object == 2) and (data == 5)) {
            MenuDataSummaryNext(outputsize, 4, animal, owner, app,
                spec, charac, gen, reason, object, data, selected);
        }
        if ((object == 3) and (data == 3)) {
            MenuDataSummaryNext(outputsize, 6, animal, owner, app,
                spec, charac, gen, reason, object, data, selected);
        }
        Clear();
        MenuDataSummaryNextText(outputsize, oldchoice, selected);
    }
    else {
        text = "Choose a sign\n";
        Outputconverter(text, outputsize);
        std::cout << text;
    }
}

```

```

        break;
    case 8:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 8);
break;
case 2:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        selected = 3;
        Clear();
        MenuDataSummaryNextText(outputsize, oldchoice, selected);
        break;
    case 2:
        selected = 4;
        Clear();
        MenuDataSummaryNextText(outputsize, oldchoice, selected);
        break;
    case 3:
        if (selected) {
            Clear();
            if (((object == 2) and (data == 1)) or ((object == 2) and
                (data == 2)) or ((object == 2) and (data == 3)) or
                ((object == 3) and (data == 4))) {
                MenuDataSummaryNext(outputsize, 5, animal, owner, app,
                    spec, charac, gen, reason, object, data, selected);
            }
            if (((object == 3) and (data == 1)) or ((object == 3) and
                (data == 2))) {
                MenuDataSummaryNext(outputsize, 3, animal, owner, app,
                    spec, charac, gen, reason, object, data, selected);
            }
            Clear();
            MenuDataSummaryNextText(outputsize, oldchoice, selected);
        }
    else {
        text = "Choose a sign\n";
        Outputconverter(text, outputsize);
        std::cout << text;
    }
}
break;

```

```

        case 4:
            break;
        default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 4);
break;
case 3:
do {
    std::cout << "Enter: ";
    std::cin >> intvalue1;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cout << "invalid input, try again\n";
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
    }
    else {
        break;
    }
} while (true);

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();
if ((object == 1) and (data == 1)) {
    text = "Number of records with Number of Animals ";
    if (selected == 1) {
        text = text + "< " + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "<";
    }
    if (selected == 2) {
        text = text + "<= " + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "<=";
    }
    if (selected == 3) {
        text = text + "= " + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        text = text + "!= " + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    if (selected == 5) {
        text = text + ">= " + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = ">=";
    }
    if (selected == 6) {
        text = text + "> " + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = ">";
    }
}

```

```

        text = text + std::to_string(SummaryOwner(owner, intvalue1,
            sign[static_cast<unsigned __int64>(data - 1)])) + '\n';
    }
    if ((object == 2) and (data == 4)) {
        text = "Number of records with Years of Age ";
        if (selected == 1) {
            text = text + "< " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "<";
        }
        if (selected == 2) {
            text = text + "<= " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "<=";
        }
        if (selected == 3) {
            text = text + "= " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {
            text = text + "!= " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "!=";
        }
        if (selected == 5) {
            text = text + ">= " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = ">=";
        }
        if (selected == 6) {
            text = text + "> " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = ">";
        }
        text = text + std::to_string(SummaryAnimal(animal, sign, strvalue1,
            strvalue2, strvalue2, intvalue1, floatvalue)) + '\n';
    }
    if ((object == 3) and (data == 1)) {
        text = "Number of records with Owner ID ";
        if (selected == 3) {
            text = text + "= " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {
            text = text + "!= " + std::to_string(intvalue1) + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "!=";
        }
        text = text + std::to_string(SummaryAppointment(app, sign,
            intvalue1, intvalue2, timevalue, strvalue1)) + '\n';
    }
    if ((object == 3) and (data == 2)) {
        text = "Number of records with Animal ID ";
        if (selected == 3) {
            text = text + "= " + std::to_string(intvalue1) + ": ";

```

```

        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        text = text + "!=" + std::to_string(intvalue1) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    text = text + std::to_string(SummaryAppointment(app, sign,
        intvalue2, intvalue1, timevalue, strvalue1)) + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
AnyKeyReturn(outputsize);
break;
case 4:
do {
    std::cout << "Enter: ";
    std::cin >> floatvalue;
    if ((std::cin.fail()) or (floatvalue <= 0)) {
        std::cin.clear();
        std::cout << "invalid input, try again\n";
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
} while (!(floatvalue) or (floatvalue <= 0));
Clear();
if ((object == 2) and (data == 5)) {
    text = "Number of records with Weight ";
    if (selected == 1) {
        text = text + "< " + std::to_string(floatvalue) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "<";
    }
    if (selected == 2) {
        text = text + "<= " + std::to_string(floatvalue) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "<=";
    }
    if (selected == 3) {
        text = text + "= " + std::to_string(floatvalue) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        text = text + "!=" + std::to_string(floatvalue) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    if (selected == 5) {
        text = text + ">= " + std::to_string(floatvalue) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = ">=";
    }
    if (selected == 6) {
        text = text + "> " + std::to_string(floatvalue) + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = ">";
    }
}

```

```

        text = text + std::to_string(SummaryAnimal(animal, sign, strvalue1,
            strvalue1, strvalue1, intvalue1, floatvalue)) + '\n';
    }
    Outputconverter(text, outputsize);
    std::cout << text;
    AnyKeyReturn(outputsize);
    break;
case 5:
    choice = 0;
    if ((object == 2) and (data == 1)) {
        TableOutputSpecies(spec, outputsize);
        choice = static_cast<unsigned short int>(spec.size() - 1) ;
    }
    if ((object == 2) and (data == 2)) {
        TableOutputCharacteristics(charac, outputsize);
        choice = static_cast<unsigned short int>(charac.size() - 1) ;
    }
    if ((object == 2) and (data == 3)) {
        TableOutputGender(gen, outputsize);
        choice = static_cast<unsigned short int>(gen.size() - 1) ;
    }
    if ((object == 3) and (data == 4)) {
        TableOutputReason(reason, outputsize);
        choice = static_cast<unsigned short int>(reason.size() - 1) ;
    }
    MenuDataSummaryNextText(outputsize, oldchoice, 0);

    do {
        std::cout << "Enter: ";
        std::cin >> intvalue1;
        if ((std::cin.fail()) or (intvalue1 > choice)) { //clear input if it was bad
            std::cin.clear();
            std::cout << "invalid input, try again\n";
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            break;
        }
    } while (true);
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    Clear();

    if ((object == 2) and (data == 1)) {
        strvalue1 = spec[intvalue1].species;
        text = "Number of records with Species ";
        if (selected == 3) {
            text = text + "= " + strvalue1 + ": ";
            sign[static_cast<unsigned __int64>(data - 1)] = "=";
        }
        if (selected == 4) {

```

```

        text = text + "!=" + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    text = text + std::to_string(SummaryAnimal(animal, sign, strvalue1,
        strvalue2, strvalue2, intvalue1, floatvalue)) + '\n';
}
if ((object == 2) and (data == 2)) {
    strvalue1 = charac[intvalue1].breed;
    text = "Number of records with Breed ";
    if (selected == 3) {
        text = text + "= " + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        text = text + "!=" + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    text = text + std::to_string(SummaryAnimal(animal, sign, strvalue2,
        strvalue1, strvalue2, intvalue2, floatvalue)) + '\n';
}
if ((object == 2) and (data == 3)) {
    strvalue1 = gen[intvalue1].gender;
    text = "Number of records with Gender ";
    if (selected == 3) {
        text = text + "= " + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        text = text + "!=" + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    text = text + std::to_string(SummaryAnimal(animal, sign, strvalue2,
        strvalue2, strvalue1, intvalue2, floatvalue)) + '\n';
}
if ((object == 3) and (data == 4)) {
    strvalue1 = reason[intvalue1].reason;
    text = "Number of records with Reason ";
    if (selected == 3) {
        text = text + "= " + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "=";
    }
    if (selected == 4) {
        text = text + "!=" + strvalue1 + ": ";
        sign[static_cast<unsigned __int64>(data - 1)] = "!=";
    }
    text = text + std::to_string(SummaryAppointment(app, sign,
        intvalue2, intvalue2, timevalue, strvalue1)) + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;

```



```

AnyKeyReturn(outputsize);
break;
case 6:
timevalue = InputDate();
localtime_s(&date, &timevalue);
Clear();
if ((object == 3) and (data == 3)) {
text = "Number of records with Date ";
if (selected == 1) {
text = text + "< " + std::to_string(date.tm_mday) + "." +
std::to_string(date.tm_mon) + "." +
std::to_string(date.tm_year + 1900) + " " +
std::to_string(date.tm_hour) + ":" +
std::to_string(date.tm_min) + ":";
sign[static_cast<unsigned __int64>(data - 1)] = "<";
}
if (selected == 2) {
text = text + "<=" + std::to_string(date.tm_mday) + "." +
std::to_string(date.tm_mon) + "." +
std::to_string(date.tm_year + 1900) + " " +
std::to_string(date.tm_hour) + ":" +
std::to_string(date.tm_min) + ":";
sign[static_cast<unsigned __int64>(data - 1)] = "<=";
}
if (selected == 3) {
text = text + "=" + std::to_string(date.tm_mday) + "." +
std::to_string(date.tm_mon) + "." +
std::to_string(date.tm_year + 1900) + " " +
std::to_string(date.tm_hour) + ":" +
std::to_string(date.tm_min) + ":";
sign[static_cast<unsigned __int64>(data - 1)] = "=";
}
if (selected == 4) {
text = text + "!=" + std::to_string(date.tm_mday) + "." +
std::to_string(date.tm_mon) + "." +
std::to_string(date.tm_year + 1900) + " " +
std::to_string(date.tm_hour) + ":" +
std::to_string(date.tm_min) + ":";
sign[static_cast<unsigned __int64>(data - 1)] = "!=";
}
if (selected == 5) {
text = text + ">=" + std::to_string(date.tm_mday) + "." +
std::to_string(date.tm_mon) + "." +
std::to_string(date.tm_year + 1900) + " " +
std::to_string(date.tm_hour) + ":" +
std::to_string(date.tm_min) + ":";
sign[static_cast<unsigned __int64>(data - 1)] = ">=";
}
if (selected == 6) {
text = text + "> " + std::to_string(date.tm_mday) + "." +

```

```

        std::to_string(date.tm_mon) + "." +
        std::to_string(date.tm_year + 1900) + " " +
        std::to_string(date.tm_hour) + ":" +
        std::to_string(date.tm_min) + " : ";
        sign[static_cast<unsigned __int64>(data - 1)] = ">";
    }
    text = text + std::to_string(SummaryAppointment(app, sign,
        intvalue2, intvalue2, timevalue, strvalue1)) + '\n';
}
Outputconverter(text, outputsize);
std::cout << text;
AnyKeyReturn(outputsize);
break;
}
}

void MenuDataSummaryText(int& outputsize, const short int choice) {

    std::string text;
    switch (choice) {
    case 1:
        text = "Select an object for summary.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Owner\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Animal\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Appointment\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "4: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 2: //handling any owner case
        text = "Select an Owner data for summary.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Number of Animals\n";
        Outputconverter(text, outputsize);
        std::cout << text;

```

```

    text = "2: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 3: //handling any animal case
    text = "Select an Animal data for summary.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Specie\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Breed\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Gender\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Years of Age\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Weight\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 4: //handling any appointment case
    text = "Select an Appointment data for summary.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Animal ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Date\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Reason\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Return\n";

```

```

    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 5:
    text = "Select an object for summary.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 6:
    text = "Select an object for summary.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 7:
    text = "Select an object for summary.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;

```

```

        text = "3: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    }
}

void MenuDataSummary(int& outputsize, const short int oldchoice,
std::vector<Animal>& animal, std::vector<Owner>& owner,
std::vector<Appointment>& app, std::vector<Species>& spec,
std::vector<Characteristics>& charac, std::vector<Gender>& gen,
std::vector<Reason>& reason) {

    short int choice;
    MenuDataSummaryText(outputsize, oldchoice);
    switch (oldchoice) {
    case 1:
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {
            case 1:
                Clear();
                MenuDataSummary(outputsize, 2, animal, owner, app, spec,
                    charac, gen, reason);
                Clear();
                MenuDataSummaryText(outputsize, oldchoice);
                break;
            case 2:
                Clear();
                MenuDataSummary(outputsize, 3, animal, owner, app, spec,
                    charac, gen, reason);
                Clear();
                MenuDataSummaryText(outputsize, oldchoice);
                break;
            case 3:
                Clear();
                MenuDataSummary(outputsize, 4, animal, owner, app, spec,
                    charac, gen, reason);
                Clear();
                MenuDataSummaryText(outputsize, oldchoice);
                break;
            case 4:
                break;
            default: std::cout << "invalid input, try again\n"; break;

```

```

    }
} while (choice != 4);
break;
case 2:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataSummaryNext(outputsize, 1, animal, owner, app, spec,
            charac, gen, reason, 1, 1, 0);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 2:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 2);
break;
case 3:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        if (spec.empty()) {
            std::cout << "No data for Species found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataSummaryNext(outputsize, 2, animal, owner, app,
                spec, charac, gen, reason, 2, 1, 0);
        }
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 2:

```

```

        Clear();
        if (charac.empty()) {
            std::cout << "No data for Breed found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataSummaryNext(outputsize, 2, animal, owner, app,
                                spec, charac, gen, reason, 2, 2, 0);
        }
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        if (gen.empty()) {
            std::cout << "No data for Gender found\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuDataSummaryNext(outputsize, 2, animal, owner, app,
                                spec, charac, gen, reason, 2, 3, 0);
        }
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataSummaryNext(outputsize, 1, animal, owner, app, spec,
                            charac, gen, reason, 2, 4, 0);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuDataSummaryNext(outputsize, 1, animal, owner, app, spec,
                            charac, gen, reason, 2, 5, 0);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 6:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 6);
break;
case 4:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;

```

```

if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    MenuDataSummaryNext(outputsize, 2, animal, owner, app, spec,
        charac, gen, reason, 3, 1, 0);
    Clear();
    MenuDataSummaryText(outputsize, oldchoice);
    break;
case 2:
    Clear();
    MenuDataSummaryNext(outputsize, 2, animal, owner, app, spec,
        charac, gen, reason, 3, 2, 0);
    Clear();
    MenuDataSummaryText(outputsize, oldchoice);
    break;
case 3:
    Clear();
    MenuDataSummaryNext(outputsize, 1, animal, owner, app, spec,
        charac, gen, reason, 3, 3, 0);
    Clear();
    MenuDataSummaryText(outputsize, oldchoice);
    break;
case 4:
    Clear();
    if (reason.empty()) {
        std::cout << "No data for Reason found\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuDataSummaryNext(outputsize, 2, animal, owner, app,
            spec, charac, gen, reason, 3, 4, 0);
    }
    Clear();
    MenuDataSummaryText(outputsize, oldchoice);
    break;
case 5:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 5);
break;
case 5:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;

```



```

    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataSummary(outputsize, 2, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataSummary(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 6:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataSummary(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataSummary(outputsize, 4, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 3:

```

```

        break;
        default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 7:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataSummary(outputsize, 3, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataSummary(outputsize, 4, animal, owner, app, spec,
            charac, gen, reason);
        Clear();
        MenuDataSummaryText(outputsize, oldchoice);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
}
}

void MenuDataShownText(int& outsize, const short int choice,
    std::vector<bool>& ownershow, std::vector<bool>& animalshow,
    std::vector<bool>& appshow) {

    std::string text;
    switch (choice) {
    case 1:
        text = "Select an object.\n\n";
        Outputconverter(text, outsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outsize);

```

```

std::cout << text;
text = "1: Owner\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Animal\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Appointment\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 2: //handling any owner case
text = "Toggle show for Owner.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Owner ID: " + bool_to_string(ownershow[0]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Full Name: " + bool_to_string(ownershow[1]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Number of Animals: " + bool_to_string(ownershow[2])
    + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 3: //handling any animal case
text = "Toggle show for Animal.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Animal ID: " + bool_to_string(animalshow[0]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Name: " + bool_to_string(animalshow[1]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Specie: " + bool_to_string(animalshow[2]) + "\n";

```

```

Outputconverter(text, outputsize);
std::cout << text;
text = "4: Breed: " + bool_to_string(animalshow[3]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Gender: " + bool_to_string(animalshow[4]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Years of Age: " + bool_to_string(animalshow[5]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Weight: " + bool_to_string(animalshow[6]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Owner ID: " + bool_to_string(animalshow[7]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "9: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 4: //handling any appointment case
text = "Toggle show for Appointment.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Appointment ID: " + bool_to_string(appshow[0]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Owner ID: " + bool_to_string(appshow[1]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Animal ID: " + bool_to_string(appshow[2]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Date: " + bool_to_string(appshow[3]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Reason: " + bool_to_string(appshow[4]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Commentary: " + bool_to_string(appshow[5]) + "\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;

```

```

case 5:
    text = "Select an object.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 6:
    text = "Select an object.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 7:
    text = "Select an object.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;

```

```

        break;
    }
}

void MenuDataShown(int& outputsize, const short int oldchoice,
std::vector<bool>& ownershow, std::vector<bool>& animalshow,
std::vector<bool>& appshow) {

    short int choice;
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow, appshow);
    switch (oldchoice) {
    case 1:
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {
            case 1:
                Clear();
                MenuDataShown(outputsize, 2, ownershow, animalshow, appshow);
                Clear();
                MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
                    appshow);
                break;
            case 2:
                Clear();
                MenuDataShown(outputsize, 3, ownershow, animalshow, appshow);
                Clear();
                MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
                    appshow);
                break;
            case 3:
                Clear();
                MenuDataShown(outputsize, 4, ownershow, animalshow, appshow);
                Clear();
                MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
                    appshow);
                break;
            case 4:
                break;
            default: std::cout << "invalid input, try again\n"; break;
            }
        } while (choice != 4);
        break;
    case 2:
        do {

```

```

choice = 0;
std::cout << "Enter your choice: ";
std::cin >> choice;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    ownershow[0] = !ownershow[0];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 2:
    ownershow[1] = !ownershow[1];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 3:
    ownershow[2] = !ownershow[2];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 4:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 4);
break;
case 3:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
case 1:
    animalshow[0] = !animalshow[0];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 2:
    animalshow[1] = !animalshow[1];

```

```

        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 3:
        animalshow[2] = !animalshow[2];
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 4:
        animalshow[3] = !animalshow[3];
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 5:
        animalshow[4] = !animalshow[4];
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 6:
        animalshow[5] = !animalshow[5];
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 7:
        animalshow[6] = !animalshow[6];
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 8:
        animalshow[7] = !animalshow[7];
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 9:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 9);
break;
case 4:
do {
    choice = 0;
    std::cout << "Enter your choice: ";

```



```

std::cin >> choice;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    appshow[0] = !appshow[0];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 2:
    appshow[1] = !appshow[1];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 3:
    appshow[2] = !appshow[2];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 4:
    appshow[3] = !appshow[3];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 5:
    appshow[4] = !appshow[4];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 6:
    appshow[5] = !appshow[5];
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 7:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 7);
break;
case 5:
do {

```

```

choice = 0;
std::cout << "Enter your choice: ";
std::cin >> choice;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    MenuDataShown(outputsize, 2, ownershow, animalshow, appshow);
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 2:
    Clear();
    MenuDataShown(outputsize, 3, ownershow, animalshow, appshow);
    Clear();
    MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
        appshow);
    break;
case 3:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 3);
break;
case 6:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
case 1:
        Clear();
        MenuDataShown(outputsize, 3, ownershow, animalshow, appshow);
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
case 2:
        Clear();
        MenuDataShown(outputsize, 4, ownershow, animalshow, appshow);
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,

```

```

        appshow);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 7:
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuDataShown(outputsize, 3, ownershow, animalshow, appshow);
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 2:
        Clear();
        MenuDataShown(outputsize, 4, ownershow, animalshow, appshow);
        Clear();
        MenuDataShownText(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        break;
    case 3:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
}
}

void MenuShowObjectsAuxiliaryText(int& outputsize) {
    std::string text;
    text = "Choose an auxiliary object to display.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;

```

```

    text = "1: Species\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Specie Breed\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Animal Gender\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Appointment Reason\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
}

void MenuShowObjectsAuxiliary(int& outputsize, char& border,
std::vector<Species>& spec, std::vector<Characteristics>& charac,
std::vector<Gender>& gen, std::vector<Reason>& reason) {

    MenuShowObjectsAuxiliaryText(outputsize);
    short int choice;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                if (spec.empty()) {
                    std::cout << "No data for Species found\n";
                }
                else {
                    TableOutputSpecies(spec, outputsize, border);
                }
                AnyKeyReturn(outputsize);
                Clear();
                MenuShowObjectsAuxiliaryText(outputsize);
                break;
            case 2:
                Clear();
                if (charac.empty()) {
                    std::cout << "No data for Specie Breed found\n";

```

```

    }
    else {
        TableOutputCharacteristics(charac, outputsize, border);
    }
    AnyKeyReturn(outputsize);
    Clear();
    MenuShowObjectsAuxiliaryText(outputsize);
    break;
case 3:
    Clear();
    if (gen.empty()) {
        std::cout << "No data for Animal Gender found\n";
    }
    else {
        TableOutputGender(gen, outputsize, border);
    }
    AnyKeyReturn(outputsize);
    Clear();
    MenuShowObjectsAuxiliaryText(outputsize);
    break;
case 4:
    Clear();
    if (reason.empty()) {
        std::cout << "No data for Appointment Reason found\n";
    }
    else {
        TableOutputReason(reason, outputsize, border);
    }
    AnyKeyReturn(outputsize);
    Clear();
    MenuShowObjectsAuxiliaryText(outputsize);
    break;
case 5:
    break;
}
} while (choice != 5);
}

void MenuShowObjectsMainText(int& outputsize, short int& choice) {

    std::string text;
    text = "Filtering and sorting applied will reset if you go back to the "
        "previous menu.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;

    switch (choice) {

```

```

case 1:
    text = "1: Show objects\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Data to be shown\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Show summary\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Filters\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Sorting\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Restore Owners\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Restore Animals\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "8: Restore Appointments\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "9: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 2:
    text = "1: Show object\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Data to be shown\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Show summary\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Filters\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Sorting\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Restore Owners\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Return\n";
    Outputconverter(text, outputsize);

```

```

std::cout << text;
break;

case 3:
    text = "1: Show object\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Data to be shown\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Show summary\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Filters\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Sorting\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Restore Animals\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

case 4:
    text = "1: Show object\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Data to be shown\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Show summary\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Filters\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Sorting\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Restore Appointments\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

case 5:

```

```

text = "1: Show objects\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Data to be shown\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Show summary\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Filters\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Sorting\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Restore Owners\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Restore Animals\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 6:
text = "1: Show objects\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Data to be shown\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Show summary\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Filters\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Sorting\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Restore Owners\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Restore Appointments\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Return\n";
Outputconverter(text, outputsize);
std::cout << text;

```



```

        break;
    case 7:
        text = "1: Show objects\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Data to be shown\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Show summary\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "4: Filters\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "5: Sorting\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "6: Restore Animals\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "7: Restore Appointments\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "8: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    }
}

void MenuShowObjectsMain(int& outputsize, char& border, short int& oldchoice,
    std::vector<Animal>& origanimal, std::vector<Owner>& origowner,
    std::vector<Appointment>& origapp, std::vector<Species>& origspec,
    std::vector<Characteristics>& origchar, std::vector<Gender>& origgen,
    std::vector<Reason>& origreason) {

    short int choice;
    std::vector<bool> ownershow;
    std::vector<bool> animalshow;
    std::vector<bool> appshow;
    std::vector<Owner> newowner;
    std::vector<Animal> newanimal;
    std::vector<Appointment> newapp;
    std::vector<unsigned short int> prio(8, 0);
    std::vector<std::string> sign(8, "dec");

    MenuShowObjectsMainText(outputsize, oldchoice);
    if (oldchoice == 1) { //swith statement does not allow new variable
        //initialisation so I use if statement for better memmory usage
        newowner = origowner;

```

```

newanimal = origanimal;
newapp = origapp;
ownershow = std::vector<bool>(3, true);
animalshow = std::vector<bool>(8, true);
appshow = std::vector<bool>(6, true);
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        TableOutputOwner(newowner, outputsize, border, ownershow);
        TableOutputAnimal(newanimal, outputsize, border, animalshow);
        TableOutputAppointment(newapp, outputsize, border, appshow);
        AnyKeyReturn(outputsize);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuDataSummary(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuDataSorting(outputsize, oldchoice, newanimal, newowner,
            newapp, prio, sign);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);

```

```

        break;
    case 6:
        newowner = origowner;
        std::cout << "Restored\n";
        break;
    case 7:
        newanimal = origanimal;
        std::cout << "Restored\n";
        break;
    case 8:
        newapp = origapp;
        std::cout << "Restored\n";
        break;
    case 9:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 9);
}
if (oldchoice == 2) {
    newowner = origowner;
    ownershow = std::vector<bool>(3, true);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                TableOutputOwner(newowner, outputsize, border, ownershow);
                AnyKeyReturn(outputsize);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 2:
                Clear();
                MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
                    appshow);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 3:
                Clear();
                MenuDataSummary(outputsize, oldchoice, newanimal, newowner,
                    newapp, origspec, origchar, origgen, origreason);
                Clear();

```

```

        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuDataSorting(outputsize, oldchoice, newanimal, newowner,
            newapp, prio, sign);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 6:
        newowner = origowner;
        std::cout << "Restored\n";
        break;
    case 7:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 7);
}
if (oldchoice == 3) {
    newanimal = origanimal;
    animalshow = std::vector<bool>(8, true);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                TableOutputAnimal(newanimal, outputsize, border, animalshow);
                AnyKeyReturn(outputsize);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 2:
                Clear();
                MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
                    appshow);
                Clear();

```

```

        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuDataSummary(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuDataSorting(outputsize, oldchoice, newanimal, newowner,
            newapp, prio, sign);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 6:
        newanimal = origanimal;
        std::cout << "Restored\n";
        break;
    case 7:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 7);
}
if (oldchoice == 4) {
    newapp = origapp;
    appshow = std::vector<bool>(6, true);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                TableOutputAppointment(newapp, outputsize, border, appshow);
                AnyKeyReturn(outputsize);
                Clear();

```

```

        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuDataSummary(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuDataSorting(outputsize, oldchoice, newanimal, newowner,
            newapp, prio, sign);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 6:
        newapp = origapp;
        std::cout << "Restored\n";
        break;
    case 7:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 7);
}
if (oldchoice == 5) {
    newowner = origowner;
    newanimal = origanimal;
    ownershow = std::vector<bool>(3, true);
    animalshow = std::vector<bool>(8, true);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad

```

```

        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        TableOutputOwner(newowner, outputsize, border, ownershow);
        TableOutputAnimal(newanimal, outputsize, border, animalshow);
        AnyKeyReturn(outputsize);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
            appshow);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuDataSummary(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
            newapp, origspec, origchar, origgen, origreason);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuDataSorting(outputsize, oldchoice, newanimal, newowner,
            newapp, prio, sign);
        Clear();
        MenuShowObjectsMainText(outputsize, oldchoice);
        break;
    case 6:
        newowner = origowner;
        std::cout << "Restored\n";
        break;
    case 7:
        newanimal = origanimal;
        std::cout << "Restored\n";
        break;
    case 8:
        break;

```

```

        default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 8);
}
if (oldchoice == 6) {
    newowner = origowner;
    newapp = origapp;
    ownershow = std::vector<bool>(3, true);
    appshow = std::vector<bool>(6, true);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                TableOutputOwner(newowner, outputsize, border, ownershow);
                TableOutputAppointment(newapp, outputsize, border, appshow);
                AnyKeyReturn(outputsize);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 2:
                Clear();
                MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
                    appshow);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 3:
                Clear();
                MenuDataSummary(outputsize, oldchoice, newanimal, newowner,
                    newapp, origspec, origchar, origgen, origreason);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 4:
                Clear();
                MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
                    newapp, origspec, origchar, origgen, origreason);
                Clear();
                MenuShowObjectsMainText(outputsize, oldchoice);
                break;
            case 5:
                Clear();
                MenuDataSorting(outputsize, oldchoice, newanimal, newowner,

```



```

        newapp, prio, sign);
    Clear();
    MenuShowObjectsMainText(outputsize, oldchoice);
    break;
case 6:
    newowner = origowner;
    std::cout << "Restored\n";
    break;
case 7:
    newapp = origapp;
    std::cout << "Restored\n";
    break;
case 8:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 8);
}
if (oldchoice == 7) {
    newanimal = origanimal;
    newapp = origapp;
    animalshow = std::vector<bool>(8, true);
    appshow = std::vector<bool>(6, true);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
        case 1:
            Clear();
            TableOutputAnimal(newanimal, outputsize, border, animalshow);
            TableOutputAppointment(newapp, outputsize, border, appshow);
            AnyKeyReturn(outputsize);
            Clear();
            MenuShowObjectsMainText(outputsize, oldchoice);
            break;
        case 2:
            Clear();
            MenuDataShown(outputsize, oldchoice, ownershow, animalshow,
                appshow);
            Clear();
            MenuShowObjectsMainText(outputsize, oldchoice);
            break;
        case 3:
            Clear();
            MenuDataSummary(outputsize, oldchoice, newanimal, newowner,

```

```

        newapp, origspec, origchar, origgen, origreason);
    Clear();
    MenuShowObjectsMainText(outputsize, oldchoice);
    break;
case 4:
    Clear();
    MenuDataFilters(outputsize, oldchoice, newanimal, newowner,
        newapp, origspec, origchar, origgen, origreason);
    Clear();
    MenuShowObjectsMainText(outputsize, oldchoice);
    break;
case 5:
    Clear();
    MenuDataSorting(outputsize, oldchoice, newanimal, newowner,
        newapp, prio, sign);
    Clear();
    MenuShowObjectsMainText(outputsize, oldchoice);
    break;
case 6:
    newanimal = origanimal;
    std::cout << "Restored\n";
    break;
case 7:
    newapp = origapp;
    std::cout << "Restored\n";
    break;
case 8:
    break;
default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 8);
}

void MenuShowObjectsText(int& outputsize) {
    std::string text;
    text = "Choose an object/s.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Show all main objects\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Show Owner data\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Show Animal data\n";
    Outputconverter(text, outputsize);

```

```

std::cout << text;
text = "4: Show Appointment data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Show Owner + Animal data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Show Owner + Appointment data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Show Animal + Appointment data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Auxiliary objects\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "9: Return\n\n";
Outputconverter(text, outputsize);
std::cout << text;
}

void MenuShowObjects(int& outputsize, char& border,
std::vector<Species>& origspec, std::vector<Characteristics>& origchar,
std::vector<Gender>& origgen, std::vector<Animal>& origanimal,
std::vector<Owner>& origowner, std::vector<Reason>& origreason,
std::vector<Appointment>& origapp) {

    MenuShowObjectsText(outputsize);
    short int choice;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice)
        {
            case 1:
                Clear();
                if (origowner.empty()) {
                    std::cout << "No data for Owner found\n";
                }
                if (origanimal.empty()) {
                    std::cout << "No data for Animal found\n";
                }
                if (origapp.empty()) {
                    std::cout << "No data for Appointment found\n";
                }
        }
    }

```

```

if (!((origowner.empty()) or (origanimal.empty()) or
(origapp.empty())) {
    MenuShowObjectsMain(outputsize, border, choice, origanimal,
        origowner, origapp, origspec, origchar, origgen,
        origreason);
}
else {
    AnyKeyReturn(outputsize);
}
Clear();
MenuShowObjectsText(outputsize);
break;
case 2:
    Clear();
    if (origowner.empty()) {
        std::cout << "No data for Owner found\n";
    }
    if (!((origowner.empty())) {
        MenuShowObjectsMain(outputsize, border, choice, origanimal,
            origowner, origapp, origspec, origchar, origgen,
            origreason);
    }
    else {
        AnyKeyReturn(outputsize);
    }
    Clear();
    MenuShowObjectsText(outputsize);
    break;
case 3:
    Clear();
    if (origanimal.empty()) {
        std::cout << "No data for Animal found\n";
    }
    if (!((origanimal.empty())) {
        MenuShowObjectsMain(outputsize, border, choice, origanimal,
            origowner, origapp, origspec, origchar, origgen,
            origreason);
    }
    else {
        AnyKeyReturn(outputsize);
    }
    Clear();
    MenuShowObjectsText(outputsize);
    break;
case 4:
    Clear();
    if (origapp.empty()) {
        std::cout << "No data for Appointment found\n";
    }
    if (!((origapp.empty())) {

```

```

        MenuShowObjectsMain(outputsize, border, choice, origanimal,
            origowner, origapp, origspec, origchar, origgen,
            origreason);
    }
    else {
        AnyKeyReturn(outputsize);
    }
    Clear();
    MenuShowObjectsText(outputsize);
    break;
case 5:
    Clear();
    if (origowner.empty()) {
        std::cout << "No data for Owner found\n";
    }
    if (origanimal.empty()) {
        std::cout << "No data for Animal found\n";
    }
    if (!((origowner.empty()) or (origanimal.empty()))) {
        MenuShowObjectsMain(outputsize, border, choice, origanimal,
            origowner, origapp, origspec, origchar, origgen,
            origreason);
    }
    else {
        AnyKeyReturn(outputsize);
    }
    Clear();
    MenuShowObjectsText(outputsize);
    break;
case 6:
    Clear();
    if (origowner.empty()) {
        std::cout << "No data for Owner found\n";
    }
    if (origapp.empty()) {
        std::cout << "No data for Appointment found\n";
    }
    if (!((origowner.empty()) or (origapp.empty()))) {
        MenuShowObjectsMain(outputsize, border, choice, origanimal,
            origowner, origapp, origspec, origchar, origgen,
            origreason);
    }
    else {
        AnyKeyReturn(outputsize);
    }
    Clear();
    MenuShowObjectsText(outputsize);
    break;
case 7:
    Clear();

```

```

        if (origanimal.empty()) {
            std::cout << "No data for Animal found\n";
        }
        if (origapp.empty()) {
            std::cout << "No data for Appointment found\n";
        }
        if (!((origanimal.empty()) or (origapp.empty()))) {
            MenuShowObjectsMain(outputsize, border, choice, origanimal,
                                origowner, origapp, origspec, origchar, origgen,
                                origreason);
        }
        else {
            AnyKeyReturn(outputsize);
        }
        Clear();
        MenuShowObjectsText(outputsize);
        break;
    case 8:
        Clear();
        MenuShowObjectsAuxiliary(outputsize, border, origspec,
                                origchar, origgen, origreason);
        Clear();
        MenuShowObjectsText(outputsize);
        break;
    case 9:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 9);
}

void MenuAddObjectsText(int& outputsize) {

    std::string text;
    text = "Choose an object to add.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Species\n";

```

```

Outputconverter(text, outputsize);
std::cout << text;
text = "5: Specie Breed\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Animal Gender\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Appointment Reason\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
}

void MenuAddOwner(int& outputsize, char& border, std::vector<Owner>& owner,
std::string& ownerfile, std::string& animalfile) {

    std::string text;
    std::string inputstr1;
    char inputchar;
    Owner newowner;
    bool check = true;
    do {
        text = "Write a natural non negative number for Owner ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newowner.owner_id;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            check = false;
            for (auto& o : owner) {
                if (o.owner_id == newowner.owner_id) {
                    text = "Owner ID must be unique\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    check = true;
                }
            }
        }
    } while (check);
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    do {
        text = "Write a string of 80 characters maximum for Owner Full"

```

```

    " Name: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::getline(std::cin, inputstr1);
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 80) {
                do {
                    text = "Your text is more than 80 characters, do you "
                        "want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) { //clear input if it was bad
                        std::cin.clear();
                        std::cin.ignore(std::numeric_limits<std::
                            streamsize>::max(), '\n');
                    }
                } else {
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                    if (std::tolower(inputchar) == 'y') {
                        StringBoulder(inputstr1, 80);
                        break;
                    }
                    if (std::tolower(inputchar) == 'n') {
                        break;
                    }
                }
            } while (true);
            if (!(inputstr1.empty())) {
                strcpy_s(newowner.full_name, inputstr1.c_str());
                break;
            }
        }
        else {
            strcpy_s(newowner.full_name, inputstr1.c_str());
            break;
        }
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

```



```

std::vector<Owner> newownervector;
newownervector.push_back(newowner);
std::vector<bool> show(3, true);
Clear();
TableOutputOwner(newownervector, outputsize, border, show);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            owner.push_back(newowner);
            SaveOwner(owner, ownerfile);
            UpdateNoAOwner(ownerfile, animalfile);
            ReadOwner(owner, ownerfile);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            break;
        }
    }
} while (true);
}

void MenuAddAnimal(int& outputsize, char& border, std::vector<Animal>& animal,
std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Species>& spec, std::vector<Characteristics>& charac,
std::vector<Gender>& gen) {

    std::string text;
    std::string inputstr1;
    unsigned int inputint;
    char inputchar;
    Animal newanimal;
    bool check = true;
    do {
        text = "Write a natural non negative number for Animal ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newanimal.animal_id;
        if (std::cin.fail()) { //clear input if it was bad

```

```

        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        check = false;
        for (auto& a : animal) {
            if (a.animal_id == newanimal.animal_id) {
                text = "Animal ID must be unique\n";
                Outputconverter(text, outputsize);
                std::cout << text;
                check = true;
            }
        }
    }
} while (check);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    text = "Write a string of 60 characters maximum for Animal Name: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::getline(std::cin, inputstr1);
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 60) {
                do {
                    text = "Your text is more than 60 characters, do you "
                        "want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) { //clear input if it was bad
                        std::cin.clear();
                        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                    }
                } else {
                    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                    if (std::tolower(inputchar) == 'y') {
                        StringBoulder(inputstr1, 60);
                        break;
                    }
                }
                if (std::tolower(inputchar) == 'n') {

```

```

        break;
    }
} while (true);
if (!(inputstr1.empty())) {
    strcpy_s(newanimal.name, inputstr1.c_str());
    break;
}
else {
    strcpy_s(newanimal.name, inputstr1.c_str());
    break;
}
}
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

TableOutputSpecies(spec, outputsize);
do {
    text = "Choose an object by writing a number after the \"*\". You can "
        " add more objects from the menu of the respective object.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> inputint;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        if (inputint < spec.size()) {
            newanimal.characteristics.species = spec[inputint];
            break;
        }
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

std::vector<Characteristics> newcharac = charac;
size_t n = 0;
while (n < newcharac.size()) {
    if (Removespaces(newcharac[n].species.species) !=
        Removespaces(newanimal.characteristics.species.species)) {
        newcharac.erase(newcharac.begin() + static_cast<__int64>(n));
    }
    else {
        n++;
    }
}

```

```

    }
    if (newcharac.empty()) {
        text = "No breeds found with this specie, add more from the respective"
            " menu.\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        AnyKeyReturn(outputsize);
        return;
    }

    TableOutputCharacteristics(newcharac, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You can"
            " add more objects from the menu of the respective object.\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputint;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            if (inputint < newcharac.size()) {
                newanimal.characteristics = newcharac[inputint];
                break;
            }
        }
    } while (true);
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    Clear();

    TableOutputGender(gen, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You can"
            " add more objects from the menu of the respective object.\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputint;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            if (inputint < gen.size()) {
                newanimal.gender = gen[inputint];
                break;
            }
        }
    } while (true);

```

```

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

do {
    text = "Write a natural non negative number for Years of Age: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> newanimal.age;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    text = "Write a real positive number for Weight: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> newanimal.weight;
    if ((std::cin.fail()) or (newanimal.weight <= 0)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    text = "Write a natural non negative number for Owner ID: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> newanimal.owner_id;
    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

std::vector<Animal> newanimalvector;
newanimalvector.push_back(newanimal);

```

```

std::vector<bool> show(8, true);
Clear();
TableOutputAnimal(newanimalvector, outputsize, border, show);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) {        //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            animal.push_back(newanimal);
            SaveAnimal(animal, animalfile);
            UpdateNoAOwner(ownerfile, animalfile);
            ReadOwner(owner, ownerfile);
            ReadAnimal(animal, animalfile);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            break;
        }
    }
} while (true);
}

void MenuAddAppointment(int& outputsize, char& border,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Reason>& reason) {

    std::string text;
    std::string inputstr1;
    unsigned int inputint;
    char inputchar;
    Appointment newapp;
    bool check = true;
    do {
        text = "Write a natural non negative number for Appointment ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newapp.appointment_id;
        if (std::cin.fail()) {        //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

```

```

    }
    else {
        check = false;
        for (auto& a : app) {
            if (a.appointment_id == newapp.appointment_id) {
                text = "Appointment ID must be unique\n";
                Outputconverter(text, outputsize);
                std::cout << text;
                check = true;
            }
        }
    }
} while (check);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    text = "Write a natural non negative number for Owner ID: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> newapp.owner_id;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

do {
    text = "Write a natural non negative number for Animal ID: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> newapp.animal_id;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        break;
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

text = "Write date and time in the following format: Day.Month.Year "
"Hour:Minute. Year cannot be less than 1900.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
newapp.date = InputDate();

```

```

TableOutputReason(reason, outputsize);
do {
    text = "Choose an object by writing a number after the \"*\". You can "
        " add more objects from the menu of the respective object.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> inputint;
    if (std::cin.fail()) {        //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        if (inputint < reason.size()) {
            newapp.reason = reason[inputint];
            break;
        }
    }
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

do {
    text = "Write a string of 160 characters maximum for Commentary: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::getline(std::cin, inputstr1);
    if (std::cin.fail()) {        //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 160) {
                do {
                    text = "Your text is more than 160 characters, do you "
                        "want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) {        //clear input if it was bad
                        std::cin.clear();
                        std::cin.ignore(std::numeric_limits<std::
                            streamsize>::max(), '\n');
                    }
                } else {
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                }
            }
        }
    }
} while (true);

```



```

        if (std::tolower(inputchar) == 'y') {
            StringBonder(inputstr1, 160);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            break;
        }
    }
} while (true);
if (!(inputstr1.empty())) {
    strcpy_s(newapp.commentary, inputstr1.c_str());
    break;
}
}
else {
    strcpy_s(newapp.commentary, inputstr1.c_str());
    break;
}
}
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

std::vector<Appointment> newappvector;
newappvector.push_back(newapp);
std::vector<bool> show(6, true);
Clear();
TableOutputAppointment(newappvector, outputsize, border, show);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            app.push_back(newapp);
            SaveAppointment(app, appfile);
            ReadAppointment(app, appfile);
            break;
        }
        if (std::tolower(inputchar) == 'n') {

```

```

        break;
    }
}
} while (true);
}

void MenuAddSpecies(int& outputsize, std::vector<Species>& spec,
std::string& specfile) {

    std::string text;
    std::string inputstr1;
    char inputchar;
    Species newspec;
    bool check = true;

    do {
        text = "Write a string of 60 characters maximum for Animal Specie: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::getline(std::cin, inputstr1);
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            inputstr1 = Removespaces(inputstr1);
            if (!(inputstr1.empty())) {
                if (inputstr1.size() > 60) {
                    do {
                        text = "Your text is more than 60 characters, do you "
                            "want to crop it?";
                        Outputconverter(text, outputsize);
                        text = text + "\nY: yes N: no\n";
                        std::cout << text;
                        std::cin >> inputchar;
                        if (std::cin.fail()) { //clear input if it was bad
                            std::cin.clear();
                            std::cin.ignore(std::numeric_limits<std::
                                streamsize>::max(), '\n');
                        }
                    }
                    else {
                        std::cin.ignore(std::numeric_limits<std::
                            streamsize>::max(), '\n');
                        if (std::tolower(inputchar) == 'y') {
                            StringBoulder(inputstr1, 60);
                            break;
                        }
                    }
                    if (std::tolower(inputchar) == 'n') {
                        break;
                    }
                }
            }
        }
    }
}

```

```

    }
    } while (true);
    if (!(inputstr1.empty())) {
        check = true;
        for (auto& s : spec) {
            if (s.species == inputstr1) {
                text = "Animal Specie must be unique\n";
                Outputconverter(text, outputsize);
                std::cout << text;
                check = false;
            }
        }
        if (check) {
            strcpy_s(newspec.species, inputstr1.c_str());
            break;
        }
    }
}
else {
    check = true;
    for (auto& s : spec) {
        if (s.species == inputstr1) {
            text = "Animal Specie must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
        }
    }
    if (check) {
        strcpy_s(newspec.species, inputstr1.c_str());
        break;
    }
}
}
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

std::vector<Species> newspecvector;
newspecvector.push_back(newspec);
TableOutputSpecies(newspecvector, outputsize);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) { //clear input if it was bad

```

```

        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            spec.push_back(newspec);
            SaveSpecies(spec, specfile);
            ReadSpecies(spec, specfile);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            break;
        }
    }
} while (true);
}

void MenuAddCharacteristics(int& outputsize,
    std::vector<Characteristics>& charac, std::string& characfile,
    std::vector<Species>& spec) {

    std::string text;
    std::string inputstr1;
    unsigned int inputint;
    char inputchar;
    Characteristics newcharac;
    bool check = true;

    TableOutputSpecies(spec, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You can"
            " add more objects from the menu of the respective object.\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputint;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            if (inputint < spec.size()) {
                newcharac.species = spec[inputint];
                break;
            }
        }
    } while (true);
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

```

```

Clear();

do {
    text = "Write a string of 60 characters maximum for Specie Breed: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::getline(std::cin, inputstr1);
    if (std::cin.fail()) {        //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 60) {
                do {
                    text = "Your text is more than 60 characters, do you "
                        "want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) {        //clear input if it was bad
                        std::cin.clear();
                        std::cin.ignore(std::numeric_limits<std::
                            streamsize>::max(), '\n');
                    }
                }
                else {
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                    if (std::tolower(inputchar) == 'y') {
                        StringBoulder(inputstr1, 60);
                        break;
                    }
                    if (std::tolower(inputchar) == 'n') {
                        break;
                    }
                }
            }
        } while (true);
        if (!(inputstr1.empty())) {
            check = true;
            for (auto& c : charac) {
                if (Removespaces(c.breed) == inputstr1) {
                    text = "Specie Breed must be unique\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    check = false;
                }
            }
            if (check) {

```

```

        strcpy_s(newcharac.breed, inputstr1.c_str());
        break;
    }
}
}
else {
    check = true;
    for (auto& c : charac) {
        if (Removespaces(c.breed) == inputstr1) {
            text = "Specie Breed must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
        }
    }
    if (check) {
        strcpy_s(newcharac.breed, inputstr1.c_str());
        break;
    }
}
}
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

std::vector<Characteristics> newcharacvector;
newcharacvector.push_back(newcharac);
TableOutputCharacteristics(newcharacvector, outputsize);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            charac.push_back(newcharac);
            SaveCharacteristics(charac, characfile);
            ReadCharacteristics(charac, characfile);
            break;
        }
    }
}

```

```

        if (std::tolower(inputchar) == 'n') {
            break;
        }
    }
} while (true);
}

void MenuAddGender(int& outputsize, std::vector<Gender>& gen,
std::string& genfile) {

    std::string text;
    std::string inputstr1;
    char inputchar;
    Gender newgen;
    bool check = true;

    do {
        text = "Write a string of 60 characters maximum for Animal Gender: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::getline(std::cin, inputstr1);
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            inputstr1 = Removespaces(inputstr1);
            if (!(inputstr1.empty())) {
                if (inputstr1.size() > 60) {
                    do {
                        text = "Your text is more than 60 characters, do you "
                            "want to crop it?";
                        Outputconverter(text, outputsize);
                        text = text + "\nY: yes N: no\n";
                        std::cout << text;
                        std::cin >> inputchar;
                        if (std::cin.fail()) { //clear input if it was bad
                            std::cin.clear();
                            std::cin.ignore(std::numeric_limits<std::
                                streamsize>::max(), '\n');
                        }
                    }
                    else {
                        std::cin.ignore(std::numeric_limits<std::
                            streamsize>::max(), '\n');
                        if (std::tolower(inputchar) == 'y') {
                            StringBoulder(inputstr1, 60);
                            break;
                        }
                    }
                    if (std::tolower(inputchar) == 'n') {
                        break;
                    }
                }
            }
        }
    }
}

```

```

    }
    }
} while (true);
if (!(inputstr1.empty())) {
    check = true;
    for (auto& g : gen) {
        if (Removespaces(g.gender) == inputstr1) {
            text = "Animal Gender must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
        }
    }
    if (check) {
        strcpy_s(newgen.gender, inputstr1.c_str());
        break;
    }
}
else {
    check = true;
    for (auto& g : gen) {
        if (Removespaces(g.gender) == inputstr1) {
            text = "Animal Gender must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
        }
    }
    if (check) {
        strcpy_s(newgen.gender, inputstr1.c_str());
        break;
    }
}
}
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

std::vector<Gender> newnewgenvector;
newnewgenvector.push_back(newgen);
TableOutputGender(newnewgenvector, outputsize);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;

```



```

        if (std::cin.fail()) {           //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::
                streamsize>::max(), '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::
                streamsize>::max(), '\n');
            if (std::tolower(inputchar) == 'y') {
                gen.push_back(newgen);
                SaveGender(gen, genfile);
                ReadGender(gen, genfile);
                break;
            }
            if (std::tolower(inputchar) == 'n') {
                break;
            }
        }
    } while (true);
}

void MenuAddReason(int& outputsize, std::vector<Reason>& reason,
    std::string& reasonfile) {

    std::string text;
    std::string inputstr1;
    char inputchar;
    Reason newreason;
    bool check = true;

    do {
        text = "Write a string of 60 characters maximum for Appointment Reason"
            ": ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::getline(std::cin, inputstr1);
        if (std::cin.fail()) {           //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            inputstr1 = Removespaces(inputstr1);
            if (!(inputstr1.empty())) {
                if (inputstr1.size() > 60) {
                    do {
                        text = "Your text is more than 60 characters, do you "
                            "want to crop it?";
                        Outputconverter(text, outputsize);
                        text = text + "\nY: yes N: no\n";
                        std::cout << text;

```

```

std::cin >> inputchar;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::
        streamsize>::max(), '\n');
}
else {
    std::cin.ignore(std::numeric_limits<std::
        streamsize>::max(), '\n');
    if (std::tolower(inputchar) == 'y') {
        StringBoulder(inputstr1, 60);
        break;
    }
    if (std::tolower(inputchar) == 'n') {
        break;
    }
}
} while (true);
if (!(inputstr1.empty())) {
    check = true;
    for (auto& r : reason) {
        if (Removespaces(r.reason) == inputstr1) {
            text = "Appointment Reason must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
        }
    }
    if (check) {
        strcpy_s(newreason.reason, inputstr1.c_str());
        break;
    }
}
else {
    check = true;
    for (auto& r : reason) {
        if (Removespaces(r.reason) == inputstr1) {
            text = "Appointment Reason must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
        }
    }
    if (check) {
        strcpy_s(newreason.reason, inputstr1.c_str());
        break;
    }
}
}

```

```

    }
}
} while (true);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
Clear();

std::vector<Reason> newreasonvector;
newreasonvector.push_back(newreason);
TableOutputReason(newreasonvector, outputsize);
do {
    text = "Confirm?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            reason.push_back(newreason);
            SaveReason(reason, reasonfile);
            ReadReason(reason, reasonfile);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            break;
        }
    }
} while (true);
}

void MenuAddObjects(int& outputsize, char& border, std::vector<Animal>& animal,
std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile) {

    MenuAddObjectsText(outputsize);
    short int choice;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
    }

```

```

if (std::cin.fail()) {    //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    MenuAddOwner(outputsize, border, owner, ownerfile, animalfile);
    Clear();
    MenuAddObjectsText(outputsize);
    break;
case 2:
    Clear();
    if ((spec.empty()) or (charac.empty()) or (gen.empty())) {
        if (spec.empty()) {
            std::cout << "No data for Species found, add it first\n";
        }
        if (charac.empty()) {
            std::cout << "No data for Species Breed found, add it"
                " first\n";
        }
        if (gen.empty()) {
            std::cout << "No data for Animal Gender found, add it"
                " first\n";
        }
        AnyKeyReturn(outputsize);
    }
    else {
        MenuAddAnimal(outputsize, border, animal, animalfile, owner,
            ownerfile, spec, charac, gen);
    }
    Clear();
    MenuAddObjectsText(outputsize);
    break;
case 3:
    Clear();
    if (reason.empty()) {
        std::cout << "No data for Appointment Reason found, add it"
            " first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuAddAppointment(outputsize, border, app, appfile,
            reason);
    }
    Clear();
    MenuAddObjectsText(outputsize);
    break;
case 4:
    Clear();

```

```

        MenuAddSpecies(outputsize, spec, specfile);
        Clear();
        MenuAddObjectsText(outputsize);
        break;
    case 5:
        Clear();
        if (spec.empty()) {
            std::cout << "No data for Species found, add it first\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuAddCharacteristics(outputsize, charac,
                                   characfile, spec);
        }
        Clear();
        MenuAddObjectsText(outputsize);
        break;
    case 6:
        Clear();
        MenuAddGender(outputsize, gen, genfile);
        Clear();
        MenuAddObjectsText(outputsize);
        break;
    case 7:
        Clear();
        MenuAddReason(outputsize, reason, reasonfile);
        Clear();
        MenuAddObjectsText(outputsize);
        break;
    case 8:
        break;
    }
} while (choice != 8);
}

void MenuRemoveObjectsNextText(int& outputsize, const short int choice) {

    std::string text;
    switch (choice) {
    case 1:
        text = "Select data to match for deletion.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Owner ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Return\n";

```

```

    Outputconverter(text, outputsize);
    std::cout << text;
    break;
case 2:
    text = "Select a data to match for deletion.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Animal ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Owner ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

case 3:
    text = "Select a data to match for deletion.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Appointment ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Owner ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Animal ID\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

case 4:
    text = "Choose an object to delete by writing a number after the"
        " \"*\". Write \"-1\" to return.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    break;

case 5:
    text = "Select how to delete Owner.\n\n";
    Outputconverter(text, outputsize);

```

```

        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Just Owner\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Owner + all animals tied to its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Owner + all appointments tied to its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "4: Owner + all animals and appointments tied to its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "5: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    }
}

void MenuRemoveObjectsNext(int& outputsize, char& border,
    const short int oldchoice, std::vector<Animal>& animal,
    std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Appointment>& app, std::string& appfile,
    std::vector<Species>& spec, std::string& specfile,
    std::vector<Characteristics>& charac, std::string& characfile,
    std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
    std::string& reasonfile) {

    unsigned short int choice;
    unsigned short int choice2;
    unsigned int intinput;
    int intinput2;
    char inputchar;
    bool check1;
    switch (oldchoice) {
    case 1:
        MenuRemoveObjectsNextText(outputsize, oldchoice);
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {

```

```

case 1:
    Clear();
    MenuRemoveObjectsNextText(outputsize, 5);
    choice2 = 0;
    do {
        std::cout << "Enter your choice: ";
        std::cin >> choice2;
        if (std::cin.fail()) {           //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

        if ((choice2 > 0) and (choice2 < 6)) {
            if (choice2 == 5) {
                break;
            }
            Clear();
            do {
                check1 = true;
                std::cout << "Enter Owner ID: ";
                std::cin >> intinput;
                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits
                        <std::streamsize>::max(), '\n');
                    std::cout << "invalid input, try again\n";
                }
            } else {
                check1 = false;
                for (auto& o : owner) {
                    if (o.owner_id == intinput) {
                        check1 = true;
                        break;
                    }
                }
            }
            if (check1) {
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
                do {
                    std::cout << "Confirm deletion?"
                        "\nY: yes N: no\n";
                    std::cin >> inputchar;
                    if (std::cin.fail()) {
                        std::cin.clear();
                        std::cin.ignore(std::
                            numeric_limits<std::streamsize>::
                                max(), '\n');
                    }
                } else {

```



```

std::cin.ignore(std::
    numeric_limits<std::streamsize>::
        max(), '\n');
if (std::tolower(inputchar)
    == 'y') {
    DeleteDataOwner(owner, app,
        animal, intinput, ownerfile,
        appfile, animalfile, choice2);
    ReadOwner(owner, ownerfile);
    check1 = false;
    MenuRemoveObjectsNextText(
        outputsize, 5);
    break;
}
if (std::tolower(inputchar)
    == 'n') {
    check1 = false;
    Clear();
    MenuRemoveObjectsNextText(
        outputsize, 5);
    break;
}
}
} while (true);
}
else {
    check1 = true;
    std::cout << "No data with that ID\n";
}
}
} while (check1);
}
else {
    std::cout << "invalid input, try again\n";
}
} while (choice2 != 5);
Clear();
MenuRemoveObjectsNextText(outputsize, oldchoice);
break;
case 2:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 2);
break;
case 2:
    MenuRemoveObjectsNextText(outputsize, oldchoice);
    do {
        choice = 0;
        std::cout << "Enter your choice: ";

```

```

std::cin >> choice;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    do {
        check1 = true;
        std::cout << "Enter Animal ID: ";
        std::cin >> intinput;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            check1 = false;
            for (auto& a : animal) {
                if (a.animal_id == intinput) {
                    check1 = true;
                    break;
                }
            }
        }
        if (check1) {
            do {
                std::cout << "Confirm deletion?"
                    "\nY: yes N: no\n";
                std::cin >> inputchar;
                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                }
            }
            else {
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
                if (std::tolower(inputchar) == 'y') {
                    DeleteDataAnimal(animal, intinput,
                        animalfile, choice);
                    ReadAnimal(animal, animalfile);
                    MenuRemoveObjectsNextText(outputsize,
                        oldchoice);
                    break;
                }
            }
            if (std::tolower(inputchar) == 'n') {

```

```

        MenuRemoveObjectsNextText(outputsize,
            oldchoice);
        break;
    }
}
} while (true);
}
else {
    check1 = true;
    std::cout << "No data with that ID\n";
}
}
} while (check1);
Clear();
MenuRemoveObjectsNextText(outputsize, oldchoice);
break;
case 2:
    Clear();
    do {
        check1 = true;
        std::cout << "Enter Owner ID: ";
        std::cin >> intinput;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            check1 = false;
            for (auto& a : animal) {
                if (a.owner_id == intinput) {
                    check1 = true;
                    break;
                }
            }
        }
        if (check1) {
            do {
                std::cout << "Confirm deletion?"
                    "\nY: yes N: no\n";
                std::cin >> inputchar;
                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                }
            }
            else {
                std::cin.ignore(std::numeric_limits<std::

```

```

        streamsize>::max(), '\n');
    if (std::tolower(inputchar) == 'y') {
        DeleteDataAnimal(animal, intinput,
            animalfile, choice);
        ReadAnimal(animal, animalfile);
        MenuRemoveObjectsNextText(outputsize,
            oldchoice);
        break;
    }
    if (std::tolower(inputchar) == 'n') {
        MenuRemoveObjectsNextText(outputsize,
            oldchoice);
        break;
    }
} while (true);
}
else {
    check1 = true;
    std::cout << "No data with that ID\n";
}
} while (check1);
Clear();
MenuRemoveObjectsNextText(outputsize, oldchoice);
break;
case 3:
    break;
default: std::cout << "invalid input, try again\n"; break;
} while (choice != 3);
break;
case 3:
    MenuRemoveObjectsNextText(outputsize, oldchoice);
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        do {
            check1 = true;
            std::cout << "Enter Appointment ID: ";
            std::cin >> intinput;
            if (std::cin.fail()) {

```

```

        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
        std::cout << "invalid input, try again\n";
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
        check1 = false;
        for (auto& a : app) {
            if (a.appointment_id == intinput) {
                check1 = true;
                break;
            }
        }
        if (check1) {
            do {
                std::cout << "Confirm deletion?"
                    "\nY: yes N: no\n";
                std::cin >> inputchar;
                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                }
            } else {
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
                if (std::tolower(inputchar) == 'y') {
                    DeleteDataAppointment(app, intinput,
                        appfile, choice);
                    ReadAppointment(app, appfile);
                    MenuRemoveObjectsNextText(outputsize,
                        oldchoice);
                    break;
                }
                if (std::tolower(inputchar) == 'n') {
                    MenuRemoveObjectsNextText(outputsize,
                        oldchoice);
                    break;
                }
            }
        } while (true);
    }
    else {
        check1 = true;
        std::cout << "No data with that ID\n";
    }
}
} while (check1);

```

```

Clear();
MenuRemoveObjectsNextText(outputsize, oldchoice);
break;
case 2:
Clear();
do {
    check1 = true;
    std::cout << "Enter Owner ID: ";
    std::cin >> intinput;
    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
        std::cout << "invalid input, try again\n";
    }
    else {
        check1 = false;
        for (auto& a : app) {
            if (a.owner_id == intinput) {
                check1 = true;
                break;
            }
        }
        if (check1) {
            do {
                std::cout << "Confirm deletion?"
                    "\nY: yes N: no\n";
                std::cin >> inputchar;
                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                }
            }
            else {
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
                if (std::tolower(inputchar) == 'y') {
                    DeleteDataAppointment(app, intinput,
                        appfile, choice);
                    ReadAppointment(app, appfile);
                    MenuRemoveObjectsNextText(outputsize,
                        oldchoice);
                    break;
                }
                if (std::tolower(inputchar) == 'n') {
                    MenuRemoveObjectsNextText(outputsize,
                        oldchoice);
                    break;
                }
            }
        }
    }
}

```

```

        } while (true);
    }
    else {
        check1 = true;
        std::cout << "No data with that ID\n";
    }
}
} while (check1);
Clear();
MenuRemoveObjectsNextText(outputsize, oldchoice);
break;
case 3:
    Clear();
    do {
        check1 = true;
        std::cout << "Enter Animal ID: ";
        std::cin >> intinput;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            check1 = false;
            for (auto& a : app) {
                if (a.animal_id == intinput) {
                    check1 = true;
                    break;
                }
            }
        }
        if (check1) {
            do {
                std::cout << "Confirm deletion?"
                    "\nY: yes N: no\n";
                std::cin >> inputchar;
                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                }
            }
            else {
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
                if (std::tolower(inputchar) == 'y') {
                    DeleteDataAppointment(app, intinput,
                        appfile, choice);
                    ReadAppointment(app, appfile);
                    MenuRemoveObjectsNextText(outputsize,
                        oldchoice);
                }
            }
        }
    }

```

```

        break;
    }
    if (std::tolower(inputchar) == 'n') {
        MenuRemoveObjectsNextText(outputsize,
            oldchoice);
        break;
    }
} while (true);
}
else {
    check1 = true;
    std::cout << "No data with that ID\n";
}
} while (check1);
Clear();
MenuRemoveObjectsNextText(outputsize, oldchoice);
break;
case 4:
    break;
default: std::cout << "invalid input, try again\n"; break;
} while (choice != 4);
break;
case 4:
    TableOutputSpecies(spec, outputsize, border);
    MenuRemoveObjectsNextText(outputsize, 5);
    do {
        intinput2 = 0;
        std::cout << "Enter your choice: ";
        std::cin >> intinput2;
        if ((std::cin.fail()) or (intinput2 < -1) or (intinput2 >=
            static_cast<int>(spec.size()))) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            std::cout << "invalid input, try again\n";
        }
    } else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
        if (intinput2 == -1) {
            break;
        }
        do {
            std::cout << "Confirm deletion of object *"
                + std::to_string(intinput2) + "?\nY: yes N: no\n";
            std::cin >> inputchar;
            if (std::cin.fail()) {

```



```

        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            spec.erase(spec.begin() + intinput2);
            SaveSpecies(spec, specfile);
            ReadSpecies(spec, specfile);
            TableOutputSpecies(spec, outputsize, border);
            MenuRemoveObjectsNextText(outputsize, 5);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            Clear();
            TableOutputSpecies(spec, outputsize, border);
            MenuRemoveObjectsNextText(outputsize, 5);
            break;
        }
    }
} while (true);
}
} while (true);
break;
case 5:
    TableOutputCharacteristics(charac, outputsize, border);
    MenuRemoveObjectsNextText(outputsize, 5);
    do {
        intinput2 = 0;
        std::cout << "Enter your choice: ";
        std::cin >> intinput2;
        if ((std::cin.fail()) or (intinput2 < -1) or (intinput2 >=
            static_cast<int>(charac.size()))) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            std::cout << "invalid input, try again\n";
        }
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
        if (intinput2 == -1) {
            break;
        }
    }
    do {
        std::cout << "Confirm deletion of object *"
            + std::to_string(intinput2) + "?\nY: yes N: no\n";
        std::cin >> inputchar;
    }

```

```

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            if (std::tolower(inputchar) == 'y') {
                charac.erase(charac.begin() + intinput2);
                SaveCharacteristics(charac, characfile);
                ReadCharacteristics(charac, characfile);
                TableOutputCharacteristics(charac, outputsize,
                    border);
                MenuRemoveObjectsNextText(outputsize, 5);
                break;
            }
            if (std::tolower(inputchar) == 'n') {
                Clear();
                TableOutputCharacteristics(charac, outputsize,
                    border);
                MenuRemoveObjectsNextText(outputsize, 5);
                break;
            }
        }
    } while (true);
} while (true);
break;
case 6:
    TableOutputGender(gen, outputsize, border);
    MenuRemoveObjectsNextText(outputsize, 5);
    do {
        intinput2 = 0;
        std::cout << "Enter your choice: ";
        std::cin >> intinput2;
        if ((std::cin.fail()) or (intinput2 < -1) or (intinput2 >=
            static_cast<int>(gen.size()))) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            if (intinput2 == -1) {
                break;
            }
            do {

```

```

        std::cout << "Confirm deletion of object *"
            + std::to_string(intinput2) + "?\nY: yes N: no\n";
        std::cin >> inputchar;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::
                max(), '\n');
            if (std::tolower(inputchar) == 'y') {
                gen.erase(gen.begin() + intinput2);
                SaveGender(gen, genfile);
                ReadGender(gen, genfile);
                TableOutputGender(gen, outputsize, border);
                MenuRemoveObjectsNextText(outputsize, 5);
                break;
            }
            if (std::tolower(inputchar) == 'n') {
                Clear();
                TableOutputGender(gen, outputsize, border);
                MenuRemoveObjectsNextText(outputsize, 5);
                break;
            }
        }
    } while (true);
}
} while (true);
break;
case 7:
    TableOutputReason(reason, outputsize, border);
    MenuRemoveObjectsNextText(outputsize, 5);
    do {
        intinput2 = 0;
        std::cout << "Enter your choice: ";
        std::cin >> intinput2;
        if ((std::cin.fail()) or (intinput2 < -1) or (intinput2 >=
            static_cast<int>(reason.size()))) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            if (intinput2 == -1) {
                break;
            }
        }
    }

```

```

do {
    std::cout << "Confirm deletion of object *"
        + std::to_string(intinput2) + "?\nY: yes N: no\n";
    std::cin >> inputchar;
    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
        if (std::tolower(inputchar) == 'y') {
            reason.erase(reason.begin() + intinput2);
            SaveReason(reason, reasonfile);
            ReadReason(reason, reasonfile);
            TableOutputReason(reason, outputsize, border);
            MenuRemoveObjectsNextText(outputsize, 5);
            break;
        }
        if (std::tolower(inputchar) == 'n') {
            Clear();
            TableOutputReason(reason, outputsize, border);
            MenuRemoveObjectsNextText(outputsize, 5);
            break;
        }
    }
} while (true);
} while (true);
break;
}
}

```

```

void MenuRemoveObjectsText(int& outputsize) {

```

```

    std::string text;
    text = "Choose an object to remove.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Appointment\n";

```

```

Outputconverter(text, outputsize);
std::cout << text;
text = "4: Species\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Specie Breed\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Animal Gender\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Appointment Reason\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
}

void MenuRemoveObjects(int& outputsize, char& border,
std::vector<Animal>& animal, std::string& animalfile,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile) {

    MenuAddObjectsText(outputsize);
    short int choice;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                if (owner.empty()) {
                    std::cout << "No data for Owner found, add it first\n";
                    AnyKeyReturn(outputsize);
                }
            else {
                MenuRemoveObjectsNext(outputsize, border, choice, animal,
                    animalfile, owner, ownerfile, app, appfile, spec, specfile,
                    charac, characfile, gen, genfile, reason, reasonfile);
            }
        }
    }
}

```

```

    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 2:
    Clear();
    if (animal.empty()) {
        std::cout << "No data for Animal found, add it first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuRemoveObjectsNext(outputsize, border, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 3:
    Clear();
    if (app.empty()) {
        std::cout << "No data for Appointment found, add it first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuRemoveObjectsNext(outputsize, border, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 4:
    Clear();
    if (spec.empty()) {
        std::cout << "No data for Species found, add it first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuRemoveObjectsNext(outputsize, border, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 5:
    Clear();
    if (charac.empty()) {
        std::cout << "No data for Specie Breed found, add it first\n";
        AnyKeyReturn(outputsize);
    }

```

```

    }
    else {
        MenuRemoveObjectsNext(outputsize, border, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 6:
    Clear();
    if (gen.empty()) {
        std::cout << "No data for Animal Gender found, add it first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuRemoveObjectsNext(outputsize, border, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 7:
    Clear();
    if (reason.empty()) {
        std::cout << "No data for Appointment Reason found, add it"
            " first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuRemoveObjectsNext(outputsize, border, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuRemoveObjectsText(outputsize);
    break;
case 8:
    break;
default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 8);
}

void MenuEditObjectsNextText(int& outputsize, const short int choice) {

    std::string text;
    switch (choice) {
    case 1:

```

```

text = "Select an Owner data edit.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Owner ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Full Name\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Return\n";
Outputconverter(text, outputsize);
std::cout << text;
break;
case 2:
text = "Select an Animal data edit.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "Menu:\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "1: Animal ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "2: Name\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Specie\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Specie Breed\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Animal Gender\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "6: Years of Age\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "7: Weight\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "8: Owner ID\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "9: Return\n";
Outputconverter(text, outputsize);
std::cout << text;

```



```

        break;

    case 3:
        text = "Select an Appointment data edit.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Appointment ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Owner ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Animal ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "4: Date\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "5: Appointment Reason\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "6: Commentary\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "7: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;

    case 4:
        text = "Choose an object to edit by writing a number after the"
            " \"*\". Write \"-1\" to return.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;

    case 5:
        text = "Select how to edit Owner ID.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Change Owner ID in Owner\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Change Owner ID in Owner and all animals tied to its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;

```

```

        text = "3: Change Owner ID in Owner and all appointments tied to"
            " its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "4: Change Owner ID in Owner and all animals and appointments"
            " tied to its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "5: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 6:
        text = "Select how to edit Animal ID.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Change Animal ID in Animal\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Change Animal ID in Animal and all appointments tied"
            " to its ID\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    case 7:
        text = "Select data to edit in Specie Breed.\n\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "Menu:\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "1: Specie\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "2: Breed\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "3: Return\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        break;
    }
}

```

```

void MenuEditOwner(int& outputsize, const int b, const int i,
    std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Animal>& animal, std::string& animalfile,
    std::vector<Appointment>& app, std::string& appfile) {
    //b: 1 - owner id 2 - full name
    /*if b == 1, i: 1 - none 2 - +animal 3 - +appointment 4 - +animal and
    appointment*/

    long int inputint;
    unsigned int oldid;
    unsigned int newid;
    bool check;
    bool check2;
    std::string text;
    std::string inputstr;
    char inputchar;
    size_t n;

    text = "Write an Owner ID for which you want to edit data. "
        "Write \"-1\" to return.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    do {
        std::cout << "Owner ID: ";
        std::cin >> inputint;
        if ((std::cin.fail()) or (inputint < -1)) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "invalid input, try again\n";
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            if (inputint == -1) {
                return;
            }
            check = false;
            n = 0;
            oldid = static_cast<unsigned int>(inputint);
            while (n < owner.size()) {
                if (owner[n].owner_id == oldid) {
                    check = true;
                    break;
                }
                else {
                    n++;
                }
            }
            if (check) {
                break;
            }
        }
    }
}

```

```

        else {
            text = "No data with this Owner ID found\n";
            Outputconverter(text, outputsize);
            std::cout << text;
        }
    }
} while (true);
if (b == 1) {
    oldid = owner[n].owner_id;
    do {
        text = "Write a natural non negative number for new Owner ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newid;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
    } else {
        check = true;
        for (auto& o : owner) {
            if (o.owner_id == newid) {
                text = "Owner ID must be unique\n";
                Outputconverter(text, outputsize);
                std::cout << text;
                check = false;
                break;
            }
        }
        if (check) {
            owner[n].owner_id = newid;
            break;
        }
    }
} while (true);
Clear();
if (i == 2) {
    for (auto& a : animal) {
        if (a.owner_id == oldid) {
            a.owner_id = owner[n].owner_id;
        }
    }
    SaveAnimal(animal, animalfile);
}
if (i == 3) {
    for (auto& a : app) {
        if (a.owner_id == oldid) {
            a.owner_id = owner[n].owner_id;
        }
    }
}

```

```

    }
    SaveAppointment(app, appfile);
}
if (i == 4) {
    for (auto& a : animal) {
        if (a.owner_id == oldid) {
            a.owner_id = owner[n].owner_id;
        }
    }
    SaveAnimal(animal, animalfile);
    for (auto& a : app) {
        if (a.owner_id == oldid) {
            a.owner_id = owner[n].owner_id;
        }
    }
    SaveAppointment(app, appfile);
}
SaveOwner(owner, ownerfile);
ReadOwner(owner, ownerfile);
}
if (b == 2) {
    check = true;
    do {
        check2 = true;
        text = "Current Full Name is " + std::string(owner[n].full_name);
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "\nWrite a string of 80 characters maximum for new Full "
            "Name\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::getline(std::cin, inputstr);
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            inputstr = Removespaces(inputstr);
            if (!(inputstr.empty())) {
                if (inputstr.size() > 80) {
                    do {
                        text = "Your text is more than 80 characters, do "
                            "you want to crop it?";
                        Outputconverter(text, outputsize);
                        text = text + "\nY: yes N: no\n";
                        std::cout << text;
                        std::cin >> inputchar;
                    } while (inputchar != 'Y' && inputchar != 'N');
                }
            }
        }
    } while (check == false || check2 == false);
}
}

```

```

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::
                streamsize>::max(), '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::
                streamsize>::max(), '\n');
            if (std::tolower(inputchar) == 'y') {
                StringBoulder(inputstr, 80);
                check2 = false;
                break;
            }
            if (std::tolower(inputchar) == 'n') {
                check2 = false;
                inputstr = "";
                break;
            }
        }
    } while (true);
    if (!(inputstr.empty())) {
        strcpy_s(owner[n].full_name, inputstr.c_str());
        break;
    }
}
else {
    strcpy_s(owner[n].full_name, inputstr.c_str());
    break;
}
}
} while (check2);
SaveOwner(owner, ownerfile);
}
}

```

```

void MenuEditAnimal(int& outputsize, const int b, const int i,
    std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Animal>& animal, std::string& animalfile,
    std::vector<Appointment>& app, std::string& appfile,
    std::vector<Species>& spec, std::vector<Characteristics>& charac,
    std::vector<Gender>& gen) {
    /*b: 1 - animal id 2 - name 3 - specie 4 - breed 5 - gender 6 - age
    7 - weight 8 - owner id*/
    //b == 1, i: 1 - none 2 - +appointment
    std::string text;
    std::string inputstr1;
    unsigned int inputint;
    long int inputlongint;
    unsigned short int inputshortint;

```

```

unsigned int oldid;
unsigned int newid;
char inputchar;
float inputfloat;
size_t n;
bool check = true;

text = "Write an Animal ID for which you want to edit data. "
      "Write \"-1\" to return.\n";
Outputconverter(text, outputsize);
std::cout << text;
do {
    std::cout << "Animal ID: ";
    std::cin >> inputlongint;
    if ((std::cin.fail()) or (inputlongint < -1)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "invalid input, try again\n";
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        if (inputlongint == -1) {
            return;
        }
        check = false;
        n = 0;
        oldid = static_cast<unsigned int>(inputlongint);
        while (n < animal.size()) {
            if (animal[n].animal_id == oldid) {
                check = true;
                break;
            }
            else {
                n++;
            }
        }
        if (check) {
            break;
        }
        else {
            text = "No data with this Animal ID found\n";
            Outputconverter(text, outputsize);
            std::cout << text;
        }
    }
} while (true);
if (b == 1) {
    oldid = animal[n].animal_id;
    do {
        text = "Write a natural non negative number for new Animal ID: ";

```

```

Outputconverter(text, outputsize);
std::cout << text;
std::cin >> newid;
if (std::cin.fail()) { //clear input if it was bad
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
}
else {
    check = true;
    for (auto& a : animal) {
        if (a.animal_id == newid) {
            text = "Animal ID must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
            break;
        }
    }
    if (check) {
        animal[n].animal_id = newid;
        break;
    }
}
} while (true);
Clear();
if (i == 2) {
    for (auto& a : app) {
        if (a.animal_id == oldid) {
            a.animal_id = animal[n].animal_id;
        }
    }
    SaveAppointment(app, appfile);
}
SaveAnimal(animal, animalfile);
ReadAnimal(animal, animalfile);
}
if (b == 2) {
    do {
        text = "Current Name is " + std::string(animal[n].name);
        Outputconverter(text, outputsize);
        std::cout << text;
        text = "\nWrite a string of 60 characters maximum for new Name\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::getline(std::cin, inputstr1);
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');

```



```

    }
    else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 60) {
                do {
                    text = "Your text is more than 60 characters, do "
                        "you want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) { //clear input if it was bad
                        std::cin.clear();
                        std::cin.ignore(std::numeric_limits<std::
                            streamsize>::max(), '\n');
                    }
                } else {
                    std::cin.ignore(std::numeric_limits<std::
                        streamsize>::max(), '\n');
                    if (std::tolower(inputchar) == 'y') {
                        StringBouncer(inputstr1, 60);
                        break;
                    }
                    if (std::tolower(inputchar) == 'n') {
                        inputstr1 = "";
                        break;
                    }
                }
            } while (true);
            if (!(inputstr1.empty())) {
                strcpy_s(animal[n].name, inputstr1.c_str());
                break;
            }
        }
        else {
            strcpy_s(animal[n].name, inputstr1.c_str());
            break;
        }
    }
} while (true);
SaveAnimal(animal, animalfile);
}

if (b == 3) {
    TableOutputSpecies(spec, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You "
            "can add more objects from the menu of the respective object.\n";
    }
}

```

```

    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> inputint;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
        if (inputint < spec.size()) {
            animal[n].characteristics.species = spec[inputint];
            break;
        }
    }
} while (true);
SaveAnimal(animal, animalfile);
}

if (b == 4) {
    TableOutputCharacteristics(charac, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You "
            "can add more objects from the menu of the respective object.\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputint;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            if (inputint < charac.size()) {
                animal[n].characteristics = charac[inputint];
                break;
            }
        }
    } while (true);
    SaveAnimal(animal, animalfile);
}

if (b == 5) {
    TableOutputGender(gen, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You "
            "can add more objects from the menu of the respective object.\n";

```

```

    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> inputint;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
    }
    else {
        if (inputint < gen.size()) {
            animal[n].gender = gen[inputint];
            break;
        }
    }
} while (true);
SaveAnimal(animal, animalfile);
}

if (b == 6) {
    do {
        text = "Write a natural non negative number for Years of Age: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputshortint;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            animal[n].age = inputshortint;
            break;
        }
    } while (true);
    SaveAnimal(animal, animalfile);
}

if (b == 7) {
    do {
        text = "Write a real positive number for Weight: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputfloat;
        if ((std::cin.fail()) or (inputfloat <= 0)) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
    }
}

```

```

        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            animal[n].weight = inputfloat;
            break;
        }
    } while (true);
    SaveAnimal(animal, animalfile);
}

if (b == 8) {
    do {
        text = "Write a natural non negative number for Owner ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputint;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            animal[n].owner_id = inputint;
            break;
        }
    } while (true);
    SaveAnimal(animal, animalfile);
    UpdateNoAOwner(ownerfile, animalfile);
    ReadOwner(owner, ownerfile);
}
}

void MenuEditAppointment(int& outputsize, const int b,
    std::vector<Appointment>& app, std::string& appfile,
    std::vector<Reason>& reason) {

    std::string text;
    std::string inputstr1;
    unsigned int inputint;
    unsigned int oldid;
    unsigned int newid;
    long int inputlongint;
    char inputchar;
    bool check = true;
    size_t n;

    text = "Write an Appointment ID for which you want to edit data. "
        "Write \"-1\" to return.\n";

```

```

Outputconverter(text, outputsize);
std::cout << text;
do {
    std::cout << "Appointment ID: ";
    std::cin >> inputlongint;
    if ((std::cin.fail()) or (inputlongint < -1)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "invalid input, try again\n";
    }
    else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        if (inputlongint == -1) {
            return;
        }
        check = false;
        n = 0;
        oldid = static_cast<unsigned int>(inputlongint);
        while (n < app.size()) {
            if (app[n].appointment_id == oldid) {
                check = true;
                break;
            }
            else {
                n++;
            }
        }
        if (check) {
            break;
        }
        else {
            text = "No data with this Appointment ID found\n";
            Outputconverter(text, outputsize);
            std::cout << text;
        }
    }
} while (true);

if (b == 1) {
    do {
        text = "Write a natural non negative number for new Appointment "
            "ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newid;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
    }
}

```

```

else {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
    check = true;
    for (auto& a : app) {
        if (a.appointment_id == newid) {
            text = "Appointment ID must be unique\n";
            Outputconverter(text, outputsize);
            std::cout << text;
            check = false;
            break;
        }
    }
    if (check) {
        app[n].appointment_id = newid;
        break;
    }
}
} while (true);
Clear();
SaveAppointment(app, appfile);
ReadAppointment(app, appfile);
}
if (b == 2) {
    do {
        text = "Write a natural non negative number for new Owner ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newid;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
    } else {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
            '\n');
        app[n].owner_id = newid;
    }
} while (true);
Clear();
SaveAppointment(app, appfile);
}

if (b == 3) {
    do {
        text = "Write a natural non negative number for new Animal ID: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> newid;
    } while (true);
    Clear();
    SaveAppointment(app, appfile);
}

```

```

        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            app[n].animal_id = newid;
        }
    } while (true);
    Clear();
    SaveAppointment(app, appfile);
}

if (b == 4) {
    text = "Write date and time in the folowing format: Day.Month.Year "
        "Hour:Minute. Year cannot be less than 1900.\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    app[n].date = InputDate();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    Clear();
    SaveAppointment(app, appfile);
}

if (b == 5) {
    TableOutputReason(reason, outputsize);
    do {
        text = "Choose an object by writing a number after the \"*\". You "
            "can add more objects from the menu of the respective object.\n";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputint;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            if (inputint < reason.size()) {
                app[n].reason = reason[inputint];
                break;
            }
        }
    } while (true);
    Clear();
    SaveAppointment(app, appfile);
}

```

```

}

if (b == 6) {
    do {
        text = "Write a string of 160 characters maximum for Commentary: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::getline(std::cin, inputstr1);
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
    } else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 160) {
                do {
                    text = "Your text is more than 160 characters, do "
                        "you want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) {
                        std::cin.clear();
                        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                    }
                } else {
                    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                    if (std::tolower(inputchar) == 'y') {
                        StringBoulder(inputstr1, 160);
                        break;
                    }
                    if (std::tolower(inputchar) == 'n') {
                        inputstr1 = "";
                        break;
                    }
                }
            }
        } while (true);
        if (!(inputstr1.empty())) {
            strcpy_s(app[n].commentary, inputstr1.c_str());
            break;
        }
    }
} else {
    strcpy_s(app[n].commentary, inputstr1.c_str());
    break;
}

```



```

    }
    }
    }
    } while (true);
    Clear();
    SaveAppointment(app, appfile);
}
}

void MenuEditCharacteristics(int& outputsize, const int b,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Species>& spec, std::vector<Animal>& animal,
std::string& animalfile) {
    //b: 1 - species 2 - breed

    std::string text;
    std::string inputstr1;
    long int inputlongint;
    char inputchar;
    bool check = true;
    size_t n;
    Characteristics newcharac;

    if (b == 1) {
        TableOutputCharacteristics(charac, outputsize);
        MenuEditObjectsNextText(outputsize, 4);
        do {
            text = "Enter your choice: ";
            Outputconverter(text, outputsize);
            std::cout << text;
            std::cin >> inputlongint;
            if ((std::cin.fail()) or (inputlongint < -1)) {
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                    '\n');
            }
            else {
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                    '\n');
                if (inputlongint == -1) {
                    return;
                }
                if (inputlongint < static_cast<long>(charac.size())) {
                    inputstr1 = charac[static_cast<unsigned __int64>
                        (inputlongint)].species.species;
                    newcharac = charac[static_cast<unsigned __int64>
                        (inputlongint)];
                    n = static_cast<size_t>(inputlongint);
                    Clear();
                    TableOutputSpecies(spec, outputsize);

```

```

text = "Choose an object to replace with by writing a "
      "number after the \"*\". You can add more objects in the "
      "respective menu. Write \"-1\" to return.\n\n";
Outputconverter(text, outputsize);
std::cout << text;
do {
    text = "Enter your choice: ";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::cin >> inputlongint;
    if ((std::cin.fail()) or (inputlongint < -1)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
        if (inputlongint == -1) {
            Clear();
            TableOutputCharacteristics(charac, outputsize);
            MenuEditObjectsNextText(outputsize, 4);
            break;
        }
        if (inputlongint < static_cast<long>(spec.size())){
            newcharac.species = spec[static_cast
                <unsigned __int64>(inputlongint)];
            check = true;

            for (auto& c : charac) {
                if ((Removespaces(c.species.species) ==
                    Removespaces(newcharac.species.species))
                    and (Removespaces(c.breed)
                        == Removespaces(newcharac.breed))) {
                    text = "Combination of Specie Breed "
                        "must be unique";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    check = false;
                    break;
                }
            }
        }
        if (check) {
            for (auto& a : animal) {
                if ((Removespaces(a.characteristics.
                    species.species) == Removespaces(
                    charac[static_cast<unsigned __int64>
                        (n)].species.species)) and
                    (Removespaces(a.characteristics.breed)
                        == Removespaces(charac[static_cast

```

```

        <unsigned __int64>(n)].breed))) {
            strcpy_s(a.characteristics.species.
                species, newcharac.species.
                species);
        }
    }
    SaveAnimal(animal, animalfile);
    ReadAnimal(animal, animalfile);
    charac[n] = newcharac;
    SaveCharacteristics(charac, characfile);
    ReadCharacteristics(charac, characfile);
    return;
}
}
} while (true);
}
} while (true);
}

if (b == 2) {
    TableOutputCharacteristics(charac, outputsize);
    MenuEditObjectsNextText(outputsize, 4);
    do {
        text = "Enter your choice: ";
        Outputconverter(text, outputsize);
        std::cout << text;
        std::cin >> inputlongint;
        if ((std::cin.fail()) or (inputlongint < -1)) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
                '\n');
            if (inputlongint == -1) {
                return;
            }
            if (inputlongint < static_cast<long>(charac.size())) {
                inputstr1 = charac[static_cast<unsigned __int64>
                    (inputlongint)].breed;
                newcharac = charac[static_cast<unsigned __int64>
                    (inputlongint)];
                n = static_cast<size_t>(inputlongint);
                Clear();
                std::cout << text;
                do {
                    text = "Current Breed is " + std::string(charac[n].

```

```

        breed);
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "\nWrite a string of 60 characters maximum for "
        "new Breed\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    std::getline(std::cin, inputstr1);
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::
            streamsize>::max(), '\n');
    }
    else {
        inputstr1 = Removespaces(inputstr1);
        if (!(inputstr1.empty())) {
            if (inputstr1.size() > 60) {
                do {
                    text = "Your text is more than 60 "
                        "characters, do you want to crop it?";
                    Outputconverter(text, outputsize);
                    text = text + "\nY: yes N: no\n";
                    std::cout << text;
                    std::cin >> inputchar;
                    if (std::cin.fail()) {
                        std::cin.clear();
                        std::cin.ignore(std::
                            numeric_limits<std::streamsize>::
                                max(), '\n');
                    }
                }
                else {
                    std::cin.ignore(std::numeric_limits
                        <std::streamsize>::max(), '\n');
                    if (std::tolower(inputchar) == 'y')
                    {
                        StringBoulder(inputstr1, 60);
                        break;
                    }
                    if (std::tolower(inputchar) == 'n')
                    {
                        inputstr1 = "";
                        break;
                    }
                }
            }
        } while (true);
    }
    else {
        strcpy_s(newcharac.breed, inputstr1.c_str()
        );
    }
}

```

```

    }
    if (!(inputstr1.empty())) {

        strcpy_s(newcharac.breed, inputstr1.c_str());
        check = true;
        for (auto& c : charac) {
            if ((Removespaces(c.species.species) ==
                Removespaces(newcharac.species.species))
                and (Removespaces(c.breed)
                    == Removespaces(newcharac.breed))) {
                text = "Combination of Specie Breed "
                    "must be unique";
                Outputconverter(text, outputsize);
                std::cout << text;
                check = false;
                break;
            }
        }
        if (check) {
            for (auto& a : animal) {
                if ((Removespaces(a.characteristics.
                    species.species) == Removespaces(
                        charac[static_cast<unsigned __int64>
                            (n)].species.species)) and
                    (Removespaces(a.characteristics.breed)
                        == Removespaces(charac[static_cast
                            <unsigned __int64>(n)].breed))) {
                    strcpy_s(a.characteristics.breed,
                        newcharac.breed);
                }
            }
            SaveAnimal(animal, animalfile);
            ReadAnimal(animal, animalfile);
            charac[n] = newcharac;
            SaveCharacteristics(charac, characfile);
            ReadCharacteristics(charac, characfile);
            return;
        }
        break;
    }

}
} while (true);
}
} while (true);
}
}

void MenuEditObjectsNext(int& outputsize, const short int oldchoice,

```

```

std::vector<Animal>& animal, std::string& animalfile,
std::vector<Owner>& owner, std::string& ownerfile,
std::vector<Appointment>& app, std::string& appfile,
std::vector<Species>& spec, std::string& specfile,
std::vector<Characteristics>& charac, std::string& characfile,
std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
std::string& reasonfile) {

```

```

    short int choice;
    short int choice2;
    long int inputint = -2;
    bool check;
    bool check2;
    std::string text;
    std::string inputstr;
    char inputchar;
    switch (oldchoice) {
    case 1:
        MenuEditObjectsNextText(outputsize, oldchoice);
        do {
            choice = 0;
            std::cout << "Enter your choice: ";
            std::cin >> choice;
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
            }
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            switch (choice) {
            case 1:
                Clear();
                MenuEditObjectsNextText(outputsize, 5);
                do {
                    choice2 = 0;
                    std::cout << "Enter your choice: ";
                    std::cin >> choice2;
                    if (std::cin.fail()) { //clear input if it was bad
                        std::cin.clear();
                    }
                    std::cin.ignore(std::numeric_limits<std::streamsize>::
                        max(), '\n');
                    switch (choice2) {
                    case 1:
                        Clear();
                        MenuEditOwner(outputsize, 1, choice2, owner,
                            ownerfile, animal, animalfile, app, appfile);
                        Clear();
                        MenuEditObjectsNextText(outputsize, 5);
                        break;
                    case 2:
                        Clear();

```

```

        MenuEditOwner(outputsize, 1, choice2, owner,
            ownerfile, animal, animalfile, app, appfile);
        Clear();
        MenuEditObjectsNextText(outputsize, 5);
        break;
    case 3:
        Clear();
        MenuEditOwner(outputsize, 1, choice2, owner,
            ownerfile, animal, animalfile, app, appfile);
        Clear();
        MenuEditObjectsNextText(outputsize, 5);
        break;
    case 4:
        Clear();
        MenuEditOwner(outputsize, 1, choice2, owner,
            ownerfile, animal, animalfile, app, appfile);
        Clear();
        MenuEditObjectsNextText(outputsize, 5);
        break;
    case 5:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice2 != 5);
Clear();
MenuEditObjectsNextText(outputsize, oldchoice);
break;
case 2:
    Clear();
    MenuEditOwner(outputsize, 2, 1, owner, ownerfile,
        animal, animalfile, app, appfile);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 3:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 3);
break;
case 2:
    MenuEditObjectsNextText(outputsize, oldchoice);
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
}
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

```

switch (choice) {
case 1:
    Clear();
    MenuEditObjectsNextText(outputsize, 6);
    do {
        choice2 = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice2;
        if (std::cin.fail()) {           //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::
            max(), '\n');
        switch (choice2) {
        case 1:
            Clear();
            MenuEditAnimal(outputsize, choice, choice2,
                owner, ownerfile, animal, animalfile, app, appfile,
                spec, charac, gen);
            Clear();
            MenuEditObjectsNextText(outputsize, 6);
            break;
        case 2:
            Clear();
            MenuEditAnimal(outputsize, choice, choice2,
                owner, ownerfile, animal, animalfile, app, appfile,
                spec, charac, gen);
            Clear();
            MenuEditObjectsNextText(outputsize, 6);
            break;
        case 3:
            break;
        default: std::cout << "invalid input, try again\n"; break;
        }
    } while (choice2 != 3);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 2:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,
        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 3:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,

```



```

        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 4:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,
        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 5:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,
        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 6:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,
        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 7:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,
        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 8:
    Clear();
    MenuEditAnimal(outputsize, choice, 1, owner,
        ownerfile, animal, animalfile, app, appfile, spec, charac,
        gen);
    Clear();
    MenuEditObjectsNextText(outputsize, oldchoice);
    break;
case 9:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 9);
break;

```

```

case 3:
MenuEditObjectsNextText(outputsize, oldchoice);
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuEditAppointment(outputsize, choice, app, appfile,
            reason);
        Clear();
        MenuEditObjectsNextText(outputsize, oldchoice);
        break;
    case 2:
        Clear();
        MenuEditAppointment(outputsize, choice, app, appfile,
            reason);
        Clear();
        MenuEditObjectsNextText(outputsize, oldchoice);
        break;
    case 3:
        Clear();
        MenuEditAppointment(outputsize, choice, app, appfile,
            reason);
        Clear();
        MenuEditObjectsNextText(outputsize, oldchoice);
        break;
    case 4:
        Clear();
        MenuEditAppointment(outputsize, choice, app, appfile,
            reason);
        Clear();
        MenuEditObjectsNextText(outputsize, oldchoice);
        break;
    case 5:
        Clear();
        MenuEditAppointment(outputsize, choice, app, appfile,
            reason);
        Clear();
        MenuEditObjectsNextText(outputsize, oldchoice);
        break;
    case 6:
        Clear();
        MenuEditAppointment(outputsize, choice, app, appfile,
            reason);

```

```

        Clear();
        MenuEditObjectsNextText(outputsize, oldchoice);
        break;
    case 7:
        break;
    default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 7);
break;
case 4:
    TableOutputSpecies(spec, outputsize);
    MenuEditObjectsNextText(outputsize, 4);
    text = "Attention! This will replace existing occurrences in all "
        "data.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    check2 = false;
    do {
        if (check2 == true) {
            check = true;
            for (auto& s : spec) {
                if (s.species == inputstr) {
                    Clear();
                    TableOutputSpecies(spec, outputsize);
                    MenuEditObjectsNextText(outputsize, 4);
                    text = "Attention! This will replace existing "
                        "occurrences in all data.\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    text = "Specie must be unique\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    check = false;
                    break;
                }
            }
        }
        if (check) {
            for (auto& c : charac) {
                if (Removespaces(c.species.species) ==
                    Removespaces(spec[static_cast<unsigned
                        __int64>(inputint)].species)) {
                    strcpy_s(c.species.species,
                        inputstr.c_str());
                }
            }
        }
        SaveCharacteristics(charac, characfile);
        ReadCharacteristics(charac, characfile);
        for (auto& a : animal) {
            if (Removespaces(a.characteristics.species.
                species) == Removespaces(spec[static_cast

```

```

        <unsigned __int64>(inputint)].
        species)) {
        strcpy_s(a.characteristics.species.
        species, inputstr.c_str());
    }
}
SaveAnimal(animal, animalfile);
ReadAnimal(animal, animalfile);
strcpy_s(spec[static_cast<unsigned __int64>
(inputint)].species, inputstr.c_str());
SaveSpecies(spec, specfile);
ReadSpecies(spec, specfile);
strcpy_s(spec[static_cast<unsigned __int64>
(inputint)].species, inputstr.c_str());
break;
}
}
check2 = false;
text = "Enter your choice: ";
Outputconverter(text, outputsize);
std::cout << text;
std::cin >> inputint;
if ((std::cin.fail()) or (inputint < -1)) {
    std::cout << (std::cin.fail());
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
}
else {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
    if (inputint == -1) {
        break;
    }
    if (inputint < static_cast<long>(spec.size())) {
        do {
            text = "Write a string of 60 characters maximum for "
                "new Specie: ";
            Outputconverter(text, outputsize);
            std::cout << text;
            std::getline(std::cin, inputstr);
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
            }
        }
        else {
            inputstr = Removespaces(inputstr);
            if (!(inputstr.empty())) {
                if (inputstr.size() > 60) {

```

```

do {
    text = "Your text is more than 60 "
        "characters, do you want to crop it?";
    Outputconverter(text, outputsize);
    text = text + "\nY: yes N: no\n";
    std::cout << text;
    std::cin >> inputchar;
    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(std::
            numeric_limits<std::streamsize>::
            max(), '\n');
    }
    else {
        std::cin.ignore(std::numeric_limits
            <std::streamsize>::max(), '\n');
        if (std::tolower(inputchar)
            == 'y') {
            StringBouncer(inputstr, 60);
            break;
        }
        if (std::tolower(inputchar)
            == 'n') {
            inputstr = "";
            break;
        }
    }
} while (true);
if (!(inputstr.empty())) {
    check2 = true;
    break;
}
}
else {
    check2 = true;
    break;
}
}
else {
    for (auto& c : charac) {
        if (Removespaces(c.species.species) ==
            Removespaces(spec[static_cast<unsigned
                __int64>(inputint)].species)) {
            strcpy_s(c.species.species,
                inputstr.c_str());
        }
    }
}
SaveCharacteristics(charac, characfile);
ReadCharacteristics(charac, characfile);
for (auto& a : animal) {

```

```

        if (Removespaces(a.characteristics.species.
species) == Removespaces(spec[static_cast
<unsigned __int64>(inputint)].
species)) {
            strcpy_s(a.characteristics.species.
species, inputstr.c_str());
        }
    }
    SaveAnimal(animal, animalfile);
    ReadAnimal(animal, animalfile);
    strcpy_s(spec[static_cast<unsigned __int64>
(inputint)].species, inputstr.c_str());
    SaveSpecies(spec, specfile);
    ReadSpecies(spec, specfile);
    break;
}
}
} while (true);
}

}
} while (true);
SaveSpecies(spec, specfile);
ReadSpecies(spec, specfile);
break;
case 5:
MenuEditObjectsNextText(outputsize, 7);
do {
    choice = 0;
    std::cout << "Enter your choice: ";
    std::cin >> choice;
    if (std::cin.fail()) { //clear input if it was bad
        std::cin.clear();
    }
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    switch (choice) {
    case 1:
        Clear();
        MenuEditCharacteristics(outputsize, choice, charac,
characfile, spec, animal, animalfile);
        Clear();
        MenuEditObjectsNextText(outputsize, 7);
        break;
    case 2:
        Clear();
        MenuEditCharacteristics(outputsize, choice, charac,
characfile, spec, animal, animalfile);
        Clear();
        MenuEditObjectsNextText(outputsize, 7);
        break;

```

```

        case 3:
            break;
        default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 3);
break;
case 6:
    TableOutputGender(gen, outputsize);
    MenuEditObjectsNextText(outputsize, 4);
    text = "Attention! This will replace existing occurrences in all "
        "data.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    check2 = false;
    do {
        if (check2 == true) {
            check = true;
            for (auto& g : gen) {
                if (Removespaces(g.gender) == inputstr) {
                    Clear();
                    TableOutputGender(gen, outputsize);
                    MenuEditObjectsNextText(outputsize, 4);
                    text = "Attention! This will not replace existing "
                        "occurrences in all data.\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    text = "Gender must be unique\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    check = false;
                    break;
                }
            }
        }
        if (check) {
            for (auto& a : animal) {
                if (Removespaces(a.gender.gender) ==
                    Removespaces(gen[static_cast<unsigned
                        __int64>(inputint)].gender)) {
                    strcpy_s(gen[static_cast<unsigned
                        __int64>(inputint)].gender, inputstr
                        .c_str());
                }
            }
            SaveAnimal(animal, animalfile);
            ReadAnimal(animal, animalfile);
            strcpy_s(gen[static_cast<unsigned __int64>
                (inputint)].gender, inputstr.c_str());
            SaveGender(gen, genfile);
            ReadGender(gen, genfile);
            strcpy_s(gen[static_cast<unsigned __int64>(inputint)]

```

```

        .gender, inputstr.c_str());
    break;
}
}
check2 = false;
text = "Enter your choice: ";
Outputconverter(text, outputsize);
std::cout << text;
std::cin >> inputint;
if ((std::cin.fail()) or (inputint < -1)) {
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
}
else {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
    if (inputint == -1) {
        break;
    }
    if (inputint < static_cast<long>(spec.size())) {
        do {
            text = "Write a string of 60 characters maximum for "
                "new Gender: ";
            Outputconverter(text, outputsize);
            std::cout << text;
            std::getline(std::cin, inputstr);
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
            }
        }
        else {
            inputstr = Removespaces(inputstr);
            if (!(inputstr.empty())) {
                if (inputstr.size() > 60) {
                    do {
                        text = "Your text is more than 60 "
                            "characters, do you want to crop it?";
                        Outputconverter(text, outputsize);
                        text = text + "\nY: yes N: no\n";
                        std::cout << text;
                        std::cin >> inputchar;
                        if (std::cin.fail()) {
                            std::cin.clear();
                            std::cin.ignore(std::
                                numeric_limits<std::streamsize>::
                                    max(), '\n');
                        }
                    }
                    else {

```



```

        std::cin.ignore(std::numeric_limits
        <std::streamsize>::max(), '\n');
        if (std::tolower(inputchar)
            == 'y') {
            StringBoulder(inputstr, 60);
            break;
        }
        if (std::tolower(inputchar)
            == 'n') {
            inputstr = "";
            break;
        }
    }
} while (true);
if (!(inputstr.empty())) {
    check2 = true;
    break;
}
}
else {
    check2 = true;
    break;
}
}
else {
    for (auto& a : animal) {
        if (Removespaces(a.gender.gender) ==
            Removespaces(gen[static_cast<unsigned
            __int64>(inputint)].gender)) {
            strcpy_s(gen[static_cast<unsigned
            __int64>(inputint)].gender, inputstr
                .c_str());
        }
    }
    SaveAnimal(animal, animalfile);
    ReadAnimal(animal, animalfile);
    strcpy_s(gen[static_cast<unsigned __int64>
        (inputint)].gender, inputstr.c_str());
    SaveGender(gen, genfile);
    ReadGender(gen, genfile);
    break;
}
}
} while (true);
}
} while (true);
SaveGender(gen, genfile);
ReadGender(gen, genfile);
break;

```

```

case 7:
    TableOutputReason(reason, outputsize);
    MenuEditObjectsNextText(outputsize, 4);
    text = "Attention! This will replace existing occurrences in all "
        "data.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    check2 = false;
    do {
        if (check2 == true) {
            check = true;
            for (auto& r : reason) {
                if (Removespaces(r.reason) == inputstr) {
                    Clear();
                    TableOutputReason(reason, outputsize);
                    MenuEditObjectsNextText(outputsize, 4);
                    text = "Attention! This will replace existing "
                        "occurrences in all data.\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    text = "Reason must be unique\n";
                    Outputconverter(text, outputsize);
                    std::cout << text;
                    check = false;
                    break;
                }
            }
        }
        if (check) {
            for (auto& a : app) {
                if (Removespaces(a.reason.reason) ==
                    Removespaces(reason[static_cast<unsigned
                        __int64>(inputint)].reason)) {
                    strcpy_s(reason[static_cast<unsigned
                        __int64>(inputint)].reason,
                        inputstr.c_str());
                }
            }
            SaveAppointment(app, appfile);
            ReadAppointment(app, appfile);
            SaveReason(reason, reasonfile);
            ReadReason(reason, reasonfile);
            strcpy_s(reason[static_cast<unsigned __int64>(inputint)]
                .reason, inputstr.c_str());
            break;
        }
    }
    check2 = false;
    text = "Enter your choice: ";
    Outputconverter(text, outputsize);
    std::cout << text;

```

```

std::cin >> inputint;
if ((std::cin.fail()) or (inputint < -1)) {
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
}
else {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
    if (inputint == -1) {
        break;
    }
    if (inputint < static_cast<long>(spec.size())) {
        do {
            text = "Write a string of 80 characters maximum for "
                "new Reason: ";
            Outputconverter(text, outputsize);
            std::cout << text;
            std::getline(std::cin, inputstr);
            if (std::cin.fail()) { //clear input if it was bad
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::
                    streamsize>::max(), '\n');
            }
        } else {
            inputstr = Removespaces(inputstr);
            if (!(inputstr.empty())) {
                if (inputstr.size() > 80) {
                    do {
                        text = "Your text is more than 80 "
                            "characters, do you want to crop it?";
                        Outputconverter(text, outputsize);
                        text = text + "\nY: yes N: no\n";
                        std::cout << text;
                        std::cin >> inputchar;
                        if (std::cin.fail()) {
                            std::cin.clear();
                            std::cin.ignore(std::
                                numeric_limits<std::streamsize>::
                                    max(), '\n');
                        }
                    } else {
                        std::cin.ignore(std::numeric_limits
                            <std::streamsize>::max(), '\n');
                        if (std::tolower(inputchar)
                            == 'y') {
                            StringBoulder(inputstr, 80);
                            break;
                        }
                    }
                    if (std::tolower(inputchar)

```

```

        == 'n') {
            inputstr = "";
            break;
        }
    }
} while (true);
if (!(inputstr.empty())) {
    check2 = true;
    break;
}
}
else {
    check2 = true;
    break;
}
}
else {
    for (auto& a : app) {
        if (Removespaces(a.reason.reason) ==
            Removespaces(reason[static_cast<unsigned
__int64>(inputint)].reason)) {
            strcpy_s(reason[static_cast<unsigned
__int64>(inputint)].reason,
                inputstr.c_str());
        }
    }
    SaveAppointment(app, appfile);
    ReadAppointment(app, appfile);
    SaveReason(reason, reasonfile);
    ReadReason(reason, reasonfile);
    break;
}
}
} while (true);
}
} while (true);
SaveReason(reason, reasonfile);
ReadReason(reason, reasonfile);
break;
}
}

```

```

void MenuEditObjectsText(int& outputsize) {

    std::string text;
    text = "Choose an object to edit.\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Attention! Editing Species, Specie Breed, Animal Gender or "

```

```

    "Appointment Reason will also change values in existing objects!\n\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Owner\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "2: Animal\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "3: Appointment\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "4: Species\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "5: Specie Breed\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "6: Animal Gender\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "7: Appointment Reason\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "8: Return\n";
    Outputconverter(text, outputsize);
    std::cout << text;
}

void MenuEditObjects(int& outputsize, std::vector<Animal>& animal,
    std::string& animalfile, std::vector<Owner>& owner, std::string& ownerfile,
    std::vector<Appointment>& app, std::string& appfile,
    std::vector<Species>& spec, std::string& specfile,
    std::vector<Characteristics>& charac, std::string& characfile,
    std::vector<Gender>& gen, std::string& genfile, std::vector<Reason>& reason,
    std::string& reasonfile) {

    MenuEditObjectsText(outputsize);
    short int choice;
    std::string text;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) {          //clear input if it was bad
            std::cin.clear();
        }
    }

```

```

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
switch (choice) {
case 1:
    Clear();
    if (owner.empty()) {
        std::cout << "No data for Owner found, add it first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuEditObjectsNext(outputsize, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuEditObjectsText(outputsize);
    break;
case 2:
    Clear();
    if ((animal.empty()) or (spec.empty()) or (charac.empty()) or
        (gen.empty())) {
        if (animal.empty()) {
            std::cout << "No data for Animal found, add it first\n";
        }
        if (spec.empty()) {
            std::cout << "No data for Species found, add it first\n";
        }
        if (charac.empty()) {
            std::cout << "No data for Species Breed found, add it"
                " first\n";
        }
        if (gen.empty()) {
            std::cout << "No data for Animal Gender found, add it"
                " first\n";
        }
        AnyKeyReturn(outputsize);
    }
    else {
        MenuEditObjectsNext(outputsize, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuEditObjectsText(outputsize);
    break;
case 3:
    Clear();
    if ((app.empty()) or (reason.empty())) {
        if (app.empty()) {
            std::cout << "No data for Appointment found, add it"
                " first\n";
        }
    }
}

```

```

    }
    if (reason.empty()) {
        std::cout << "No data for Appointment Reason found, add it"
            " first\n";
    }
    AnyKeyReturn(outputsize);
}
else {
    MenuEditObjectsNext(outputsize, choice, animal,
        animalfile, owner, ownerfile, app, appfile, spec, specfile,
        charac, characfile, gen, genfile, reason, reasonfile);
}
Clear();
MenuEditObjectsText(outputsize);
break;
case 4:
    Clear();
    if (spec.empty()) {
        std::cout << "No data for Species found, add it"
            " first\n";
        AnyKeyReturn(outputsize);
    }
    else {
        MenuEditObjectsNext(outputsize, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();
    MenuEditObjectsText(outputsize);
    break;
case 5:
    Clear();
    if ((spec.empty()) or (charac.empty())) {
        if (spec.empty()) {
            std::cout << "No data for Species found, add it first\n";
            AnyKeyReturn(outputsize);
        }
        if (charac.empty()) {
            std::cout << "No data for Specie Breed found, add it"
                " first\n";
        }
        AnyKeyReturn(outputsize);
    }
    else {
        MenuEditObjectsNext(outputsize, choice, animal,
            animalfile, owner, ownerfile, app, appfile, spec, specfile,
            charac, characfile, gen, genfile, reason, reasonfile);
    }
    Clear();

```

```

        MenuEditObjectsText(outputsize);
        break;
    case 6:
        Clear();
        if (gen.empty()) {
            std::cout << "No data for Animal Gender found, add it"
                " first\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuEditObjectsNext(outputsize, choice, animal,
                animalfile, owner, ownerfile, app, appfile, spec, specfile,
                charac, characfile, gen, genfile, reason, reasonfile);
        }
        Clear();
        MenuEditObjectsText(outputsize);
        break;
    case 7:
        Clear();
        if (reason.empty()) {
            std::cout << "No data for Appointment Reason found, add it"
                " first\n";
            AnyKeyReturn(outputsize);
        }
        else {
            MenuEditObjectsNext(outputsize, choice, animal,
                animalfile, owner, ownerfile, app, appfile, spec, specfile,
                charac, characfile, gen, genfile, reason, reasonfile);
        }
        Clear();
        MenuEditObjectsText(outputsize);
        break;
    case 8:
        break;
    default: std::cout << "invalid input, try again\n"; break;
    }
} while (choice != 8);
}

void MenuText(int& outputsize) {
    std::string text;
    text = "Welcome to Vet Clinic database.\n\n";
    Outputconverter(text, outputsize); //making sure it works with custom
    //output size
    std::cout << text;
    text = "Menu:\n";
    Outputconverter(text, outputsize);
    std::cout << text;
    text = "1: Show data\n";
    Outputconverter(text, outputsize);

```



```

std::cout << text;
text = "2: Add data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "3: Remove data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "4: Edit data\n";
Outputconverter(text, outputsize);
std::cout << text;
text = "5: Exit program\n\n";
Outputconverter(text, outputsize);
std::cout << text;
}

```

```

void Menu(int& outputsize, char& border, std::string& specfile,
std::string& characfile, std::string& genfile, std::string& animalfile,
std::string& ownerfile, std::string& reasonfile, std::string& appfile) {

```

```

    std::vector<Species> origspec;
    std::vector<Characteristics> origchar;
    std::vector<Gender> origgen;
    std::vector<Animal> origanimal;
    std::vector<Owner> origowner;
    std::vector<Reason> origreason;
    std::vector<Appointment> origapp;

```

```

    MenuText(outputsize);
    FilesInitialise(origspec, specfile, origchar, characfile, origgen, genfile,
        origanimal, animalfile, origowner, ownerfile, origreason, reasonfile,
        origapp, appfile);

```

```

    short int choice;
    do {
        choice = 0;
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        if (std::cin.fail()) { //clear input if it was bad
            std::cin.clear();
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        switch (choice) {
            case 1:
                Clear();
                MenuShowObjects(outputsize, border, origspec, origchar,
                    origgen, origanimal, origowner, origreason, origapp);
                Clear();
                MenuText(outputsize);
                break;
            case 2:

```

```

    Clear();
    MenuAddObjects(outputsize, border, origanimal, animalfile,
        origowner, ownerfile, origapp, appfile, origspec, specfile,
        origchar, characfile, origgen, genfile, origreason,
        reasonfile);
    Clear();
    MenuText(outputsize);
    break;
case 3:
    Clear();
    MenuRemoveObjects(outputsize, border, origanimal, animalfile,
        origowner, ownerfile, origapp, appfile, origspec, specfile,
        origchar, characfile, origgen, genfile, origreason,
        reasonfile);
    Clear();
    MenuText(outputsize);
    break;
case 4:
    Clear();
    MenuEditObjects(outputsize, origanimal, animalfile, origowner,
        ownerfile, origapp, appfile, origspec, specfile, origchar,
        characfile, origgen, genfile, origreason, reasonfile);
    Clear();
    MenuText(outputsize);
    break;
case 5:
    break;
default: std::cout << "invalid input, try again\n"; break;
}
} while (choice != 5);
}

```