On 15th June 2024

Project Title: Animal Shelter Management System

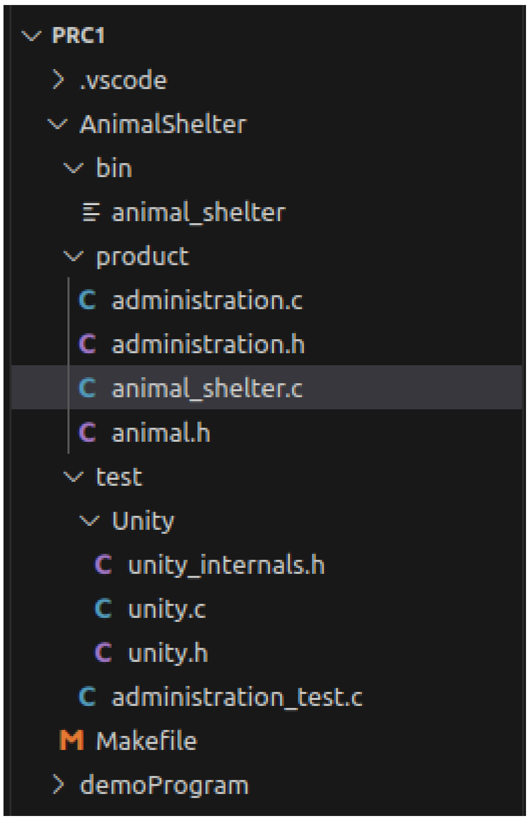
Name: Fabiola INGABIRE
Email: fabiolaingabire02@gmail.com
ROBOTIC TEAM

Introduction

Purpose: The Animal Shelter Management System is a console-based application designed to manage animals in a shelter. It allows users to add, remove, sort, and find animals.

File Structure

- **animal_shelter.c:** Main file containing the user interface and main logic.
- **administration.c: I**mplements functions for manipulating animal data (add, remove, sort, find).
- **administration.h:** Header file declaring the functions used in `administration.c`.
- **administration_test.c:** Contains unit tests for the functions in `administration.c`.
- **Makefile:** Script to build the project and run tests.
- **Unity/:** Directory containing the Unity testing framework.

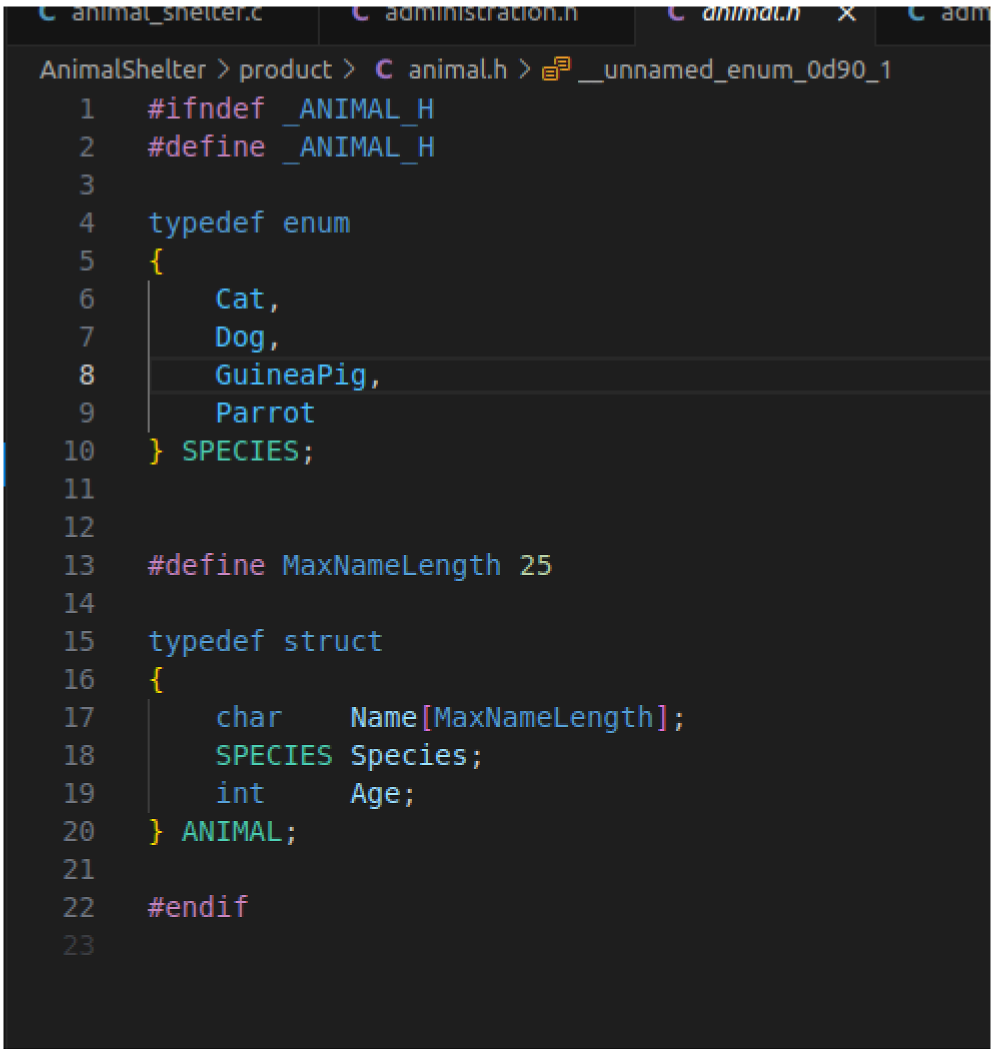


```
PRC1
  .vscode
  AnimalShelter
    bin
      animal_shelter
    product
      administration.c
      administration.h
      animal_shelter.c
      animal.h
    test
      Unity
        unity_internals.h
        unity.c
        unity.h
      administration_test.c
    Makefile
  demoProgram
```

Code Explanation

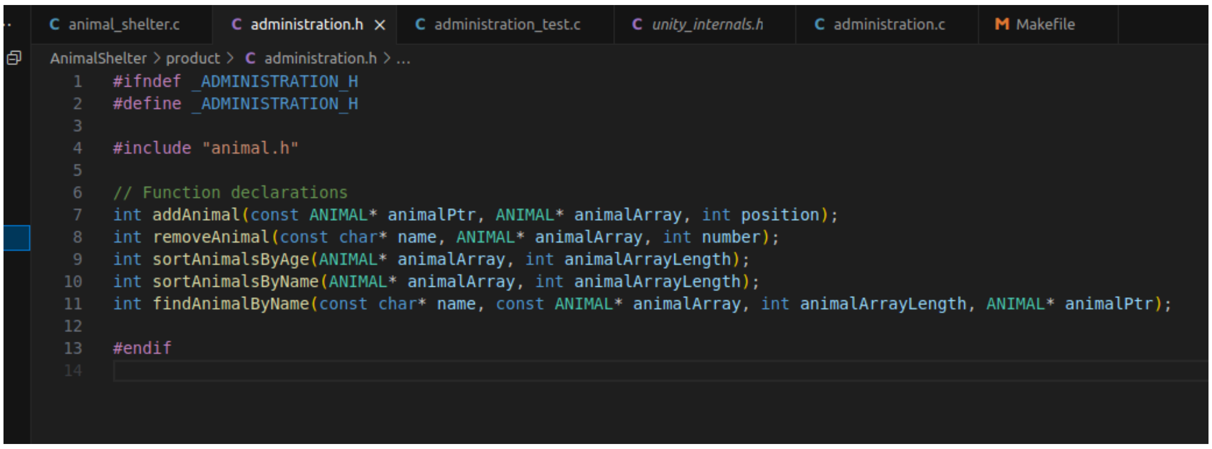
Header Files to be included in the code to: animal.h and administration.h

Functionality : Header files in C programming serve several key purposes: they provide function declarations to inform the compiler about the functions' names, return types, and parameters; they define custom data types using **`typedef`** or **`struct`;** they set constants and macros for shared use across multiple source files; they include guards to prevent multiple inclusions of the same header file, avoiding compilation errors; and they enhance code organisation, readability, maintainability, and reusability by separating the interface (declarations) from the implementation (definitions). Additionally, header files facilitate code reuse and dependency management, ensuring consistency and reducing code duplication, which is essential for maintaining a clean, modular, and efficient codebase.



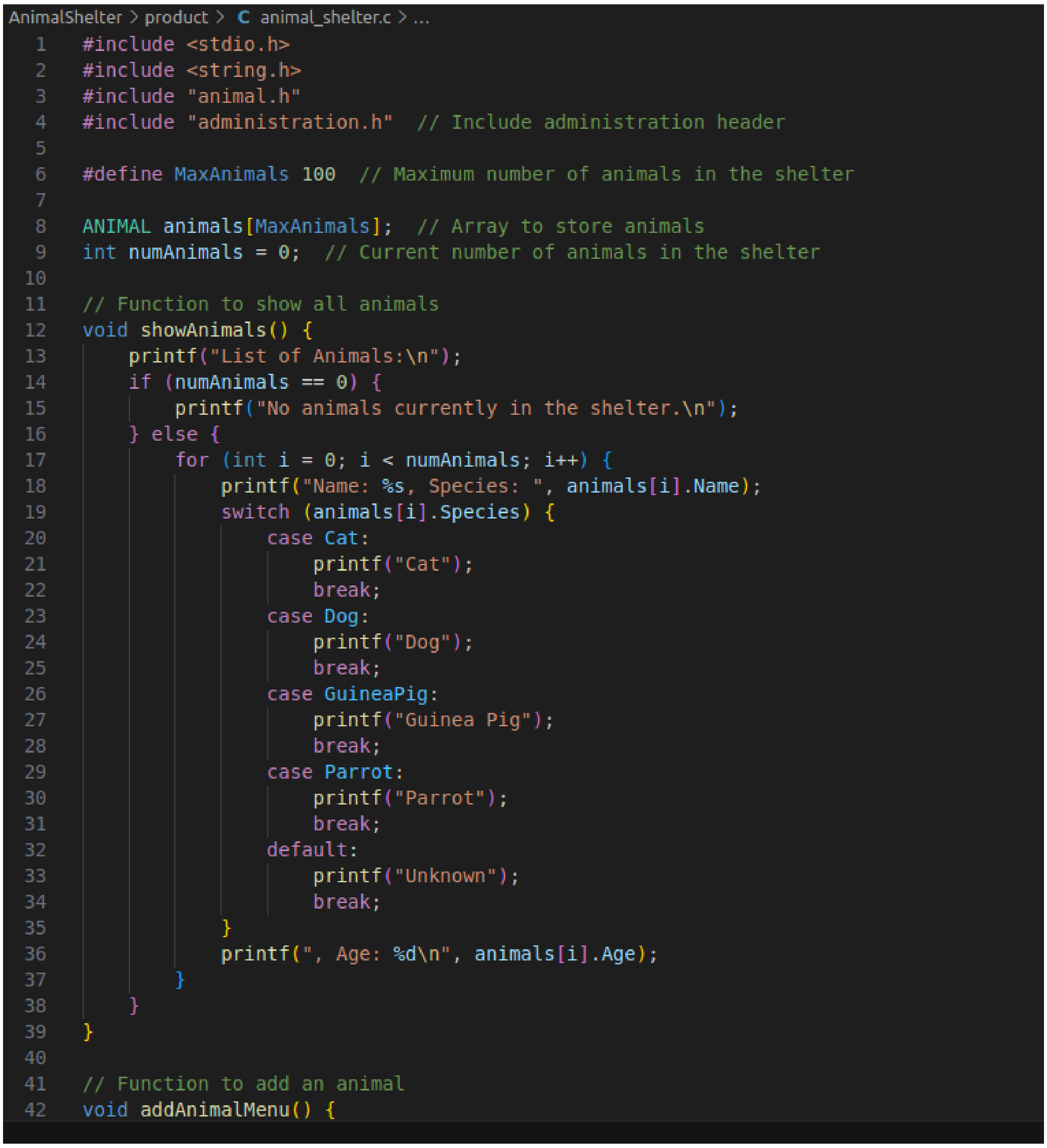
```c
#ifndef _ANIMAL_H
#define _ANIMAL_H

typedef enum
{
    Cat,
    Dog,
    GuineaPig,
    Parrot
} SPECIES;


#define MaxNameLength 25

typedef struct
{
    char    Name[MaxNameLength];
    SPECIES Species;
    int     Age;
} ANIMAL;

#endif
```

administration .h

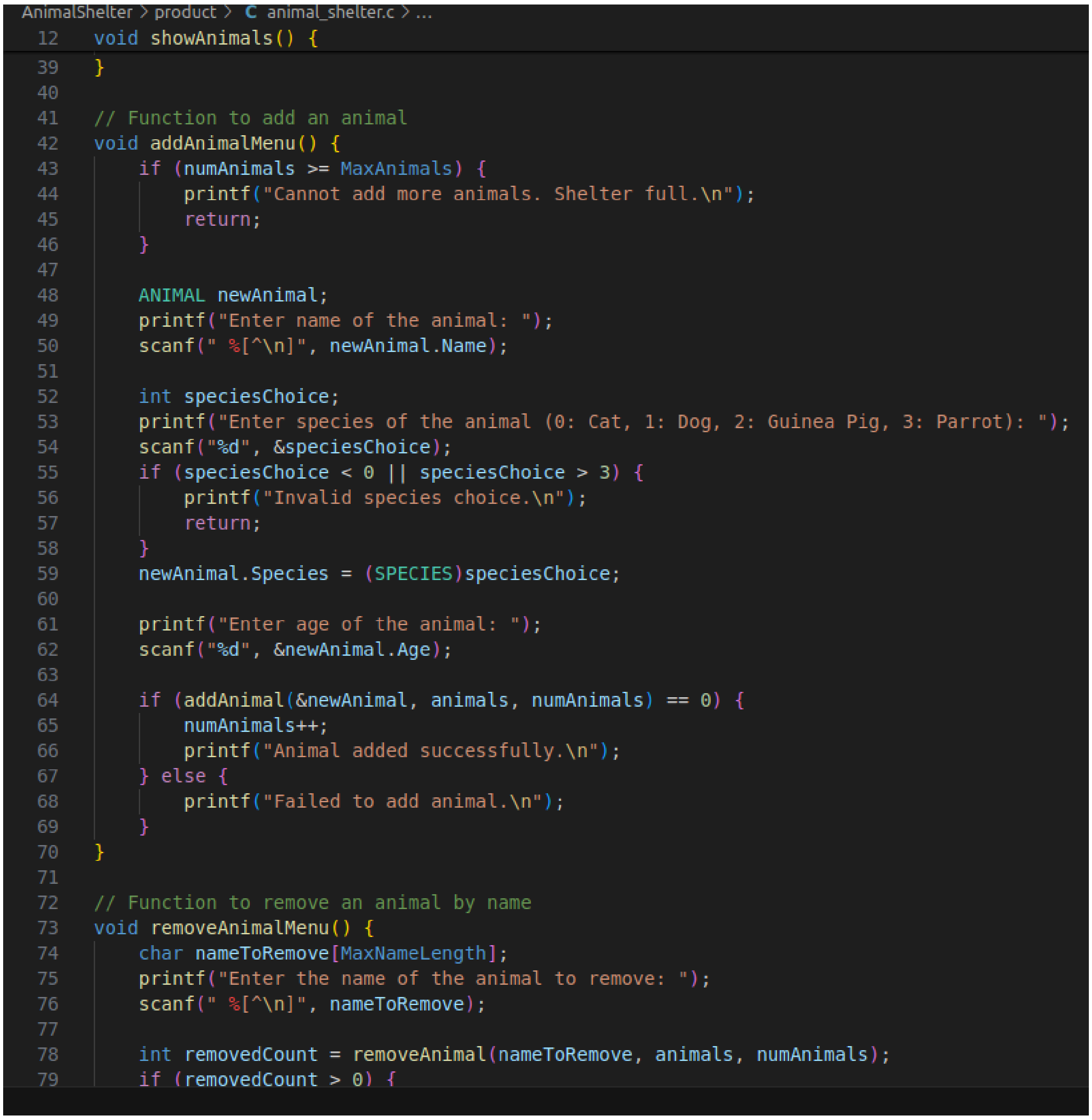


```c
#ifndef _ADMINISTRATION_H
#define _ADMINISTRATION_H

#include "animal.h"

// Function declarations
int addAnimal(const ANIMAL* animalPtr, ANIMAL* animalArray, int position);
int removeAnimal(const char* name, ANIMAL* animalArray, int number);
int sortAnimalsByAge(ANIMAL* animalArray, int animalArrayLength);
int sortAnimalsByName(ANIMAL* animalArray, int animalArrayLength);
int findAnimalByName(const char* name, const ANIMAL* animalArray, int animalArrayLength, ANIMAL* animalPtr);

#endif
```

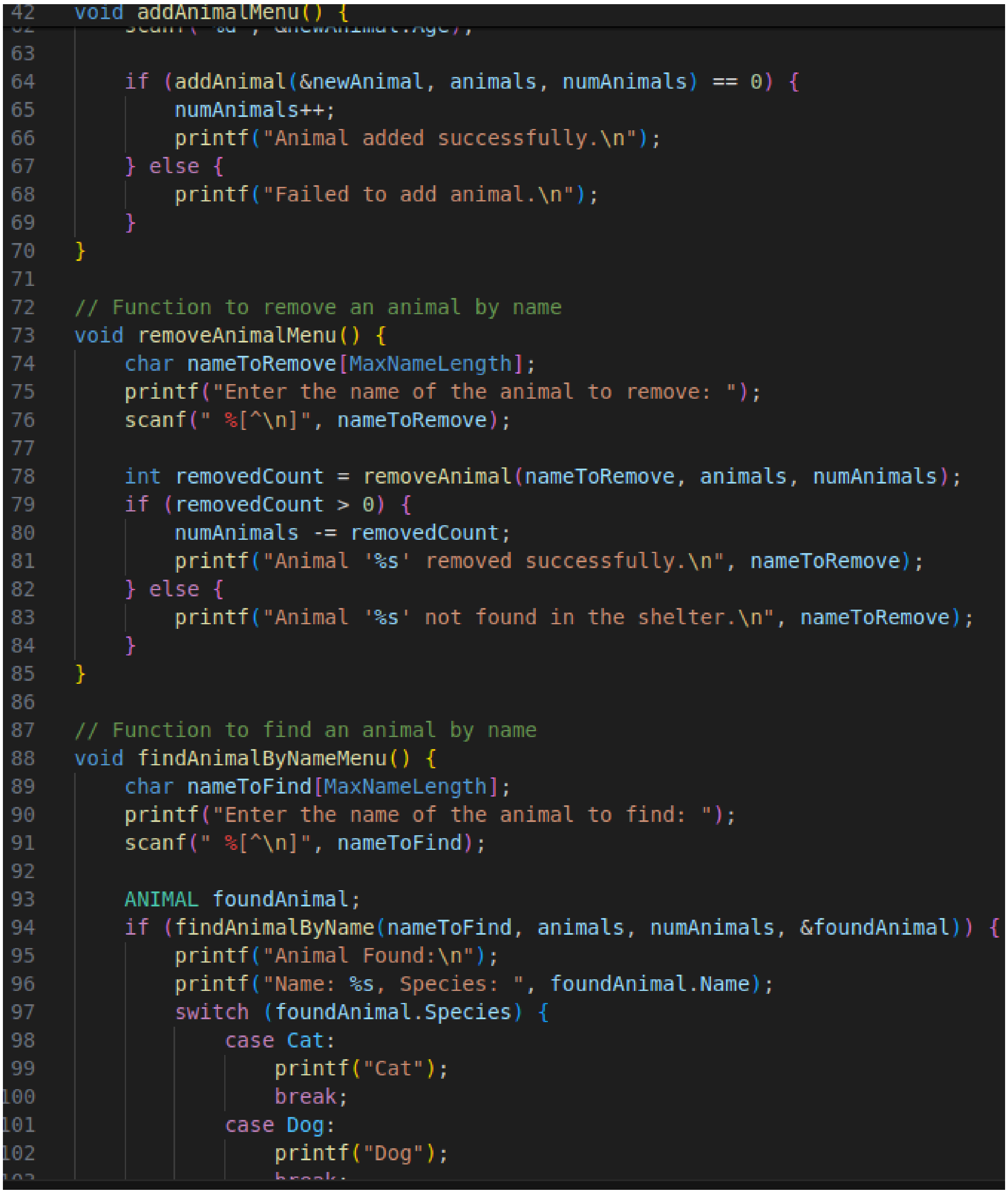
Code for: Animal_shelter.c



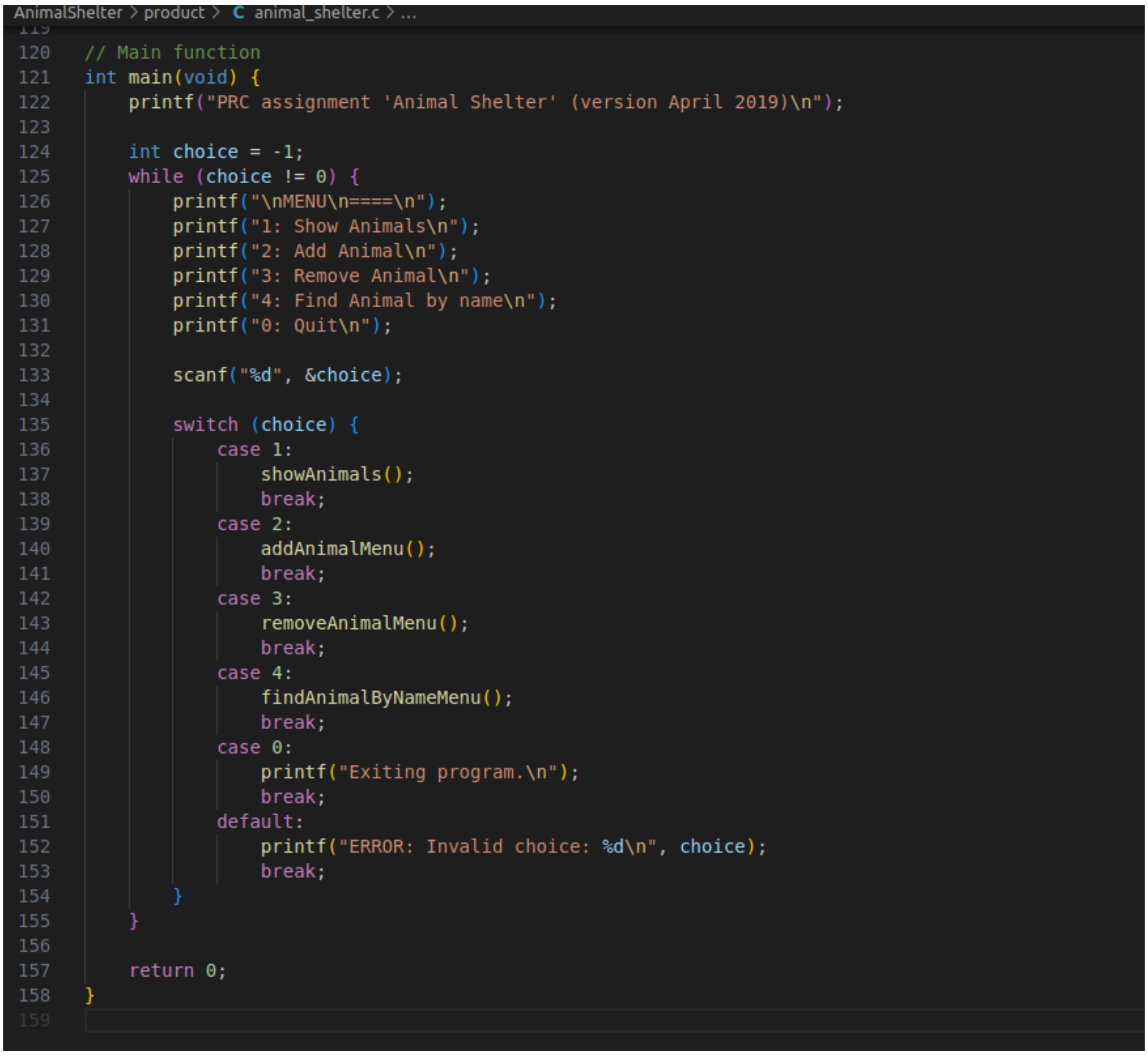
```c
#include <stdio.h>
#include <string.h>
#include "animal.h"
#include "administration.h"  // Include administration header

#define MaxAnimals 100  // Maximum number of animals in the shelter

ANIMAL animals[MaxAnimals];  // Array to store animals
int numAnimals = 0;  // Current number of animals in the shelter

// Function to show all animals
void showAnimals() {
    printf("List of Animals:\n");
    if (numAnimals == 0) {
        printf("No animals currently in the shelter.\n");
    } else {
        for (int i = 0; i < numAnimals; i++) {
            printf("Name: %s, Species: ", animals[i].Name);
            switch (animals[i].Species) {
                case Cat:
                    printf("Cat");
                    break;
                case Dog:
                    printf("Dog");
                    break;
                case GuineaPig:
                    printf("Guinea Pig");
                    break;
                case Parrot:
                    printf("Parrot");
                    break;
                default:
                    printf("Unknown");
                    break;
            }
            printf(", Age: %d\n", animals[i].Age);
        }
    }
}

// Function to add an animal
void addAnimalMenu() {
```


AnimalShelter > product > C animal_shelter.c > ...

```c
void showAnimals() {
}

// Function to add an animal
void addAnimalMenu() {
    if (numAnimals >= MaxAnimals) {
        printf("Cannot add more animals. Shelter full.\n");
        return;
    }

    ANIMAL newAnimal;
    printf("Enter name of the animal: ");
    scanf(" %[^\n]", newAnimal.Name);

    int speciesChoice;
    printf("Enter species of the animal (0: Cat, 1: Dog, 2: Guinea Pig, 3: Parrot): ");
    scanf("%d", &speciesChoice);
    if (speciesChoice < 0 || speciesChoice > 3) {
        printf("Invalid species choice.\n");
        return;
    }
    newAnimal.Species = (SPECIES)speciesChoice;

    printf("Enter age of the animal: ");
    scanf("%d", &newAnimal.Age);

    if (addAnimal(&newAnimal, animals, numAnimals) == 0) {
        numAnimals++;
        printf("Animal added successfully.\n");
    } else {
        printf("Failed to add animal.\n");
    }
}

// Function to remove an animal by name
void removeAnimalMenu() {
    char nameToRemove[MaxNameLength];
    printf("Enter the name of the animal to remove: ");
    scanf(" %[^\n]", nameToRemove);

    int removedCount = removeAnimal(nameToRemove, animals, numAnimals);
    if (removedCount > 0) {
```



```c
void addAnimalMenu() {

    if (addAnimal(&newAnimal, animals, numAnimals) == 0) {
        numAnimals++;
        printf("Animal added successfully.\n");
    } else {
        printf("Failed to add animal.\n");
    }
}

// Function to remove an animal by name
void removeAnimalMenu() {
    char nameToRemove[MaxNameLength];
    printf("Enter the name of the animal to remove: ");
    scanf(" %[^\n]", nameToRemove);

    int removedCount = removeAnimal(nameToRemove, animals, numAnimals);
    if (removedCount > 0) {
        numAnimals -= removedCount;
        printf("Animal '%s' removed successfully.\n", nameToRemove);
    } else {
        printf("Animal '%s' not found in the shelter.\n", nameToRemove);
    }
}

// Function to find an animal by name
void findAnimalByNameMenu() {
    char nameToFind[MaxNameLength];
    printf("Enter the name of the animal to find: ");
    scanf(" %[^\n]", nameToFind);

    ANIMAL foundAnimal;
    if (findAnimalByName(nameToFind, animals, numAnimals, &foundAnimal)) {
        printf("Animal Found:\n");
        printf("Name: %s, Species: ", foundAnimal.Name);
        switch (foundAnimal.Species) {
            case Cat:
                printf("Cat");
                break;
            case Dog:
                printf("Dog");
```



AnimalShelter > product > C animal_shelter.c > ...

```c
// Main function
int main(void) {
    printf("PRC assignment 'Animal Shelter' (version April 2019)\n");

    int choice = -1;
    while (choice != 0) {
        printf("\nMENU\n====\n");
        printf("1: Show Animals\n");
        printf("2: Add Animal\n");
        printf("3: Remove Animal\n");
        printf("4: Find Animal by name\n");
        printf("0: Quit\n");

        scanf("%d", &choice);

        switch (choice) {
            case 1:
                showAnimals();
                break;
            case 2:
                addAnimalMenu();
                break;
            case 3:
                removeAnimalMenu();
                break;
            case 4:
                findAnimalByNameMenu();
                break;
            case 0:
                printf("Exiting program.\n");
                break;
            default:
                printf("ERROR: Invalid choice: %d\n", choice);
                break;
        }
    }

    return 0;
}
```

Purpose: Provides the user interface for managing animals.

Key Functions:

1. showAnimals()

- Purpose: Displays all animals currently in the shelter.

- Description: Iterates through the `animals` array and prints the details of each animal.

```c
void showAnimals() {
        // Code to display animals
}
```

2. addAnimal()
  - Purpose: Adds a new animal to the shelter.

  - Description: Prompts the user for animal details and adds the animal to the `animals` array if there's space.

```c
void addAnimal() {
      // Code to add animal
}
```

3. removeAnimal()
 - Purpose: Removes an animal from the shelter by name.

  - Description: Prompts the user for the animal name and removes the animal from the `animals` array.

```c
void removeAnimal() {
      // Code to remove animal
}
```

4. findAnimalByName()
  - **Purpose:** Finds and displays an animal by name.

  - Description: Prompts the user for the animal name and searches the `animals` array for a match.

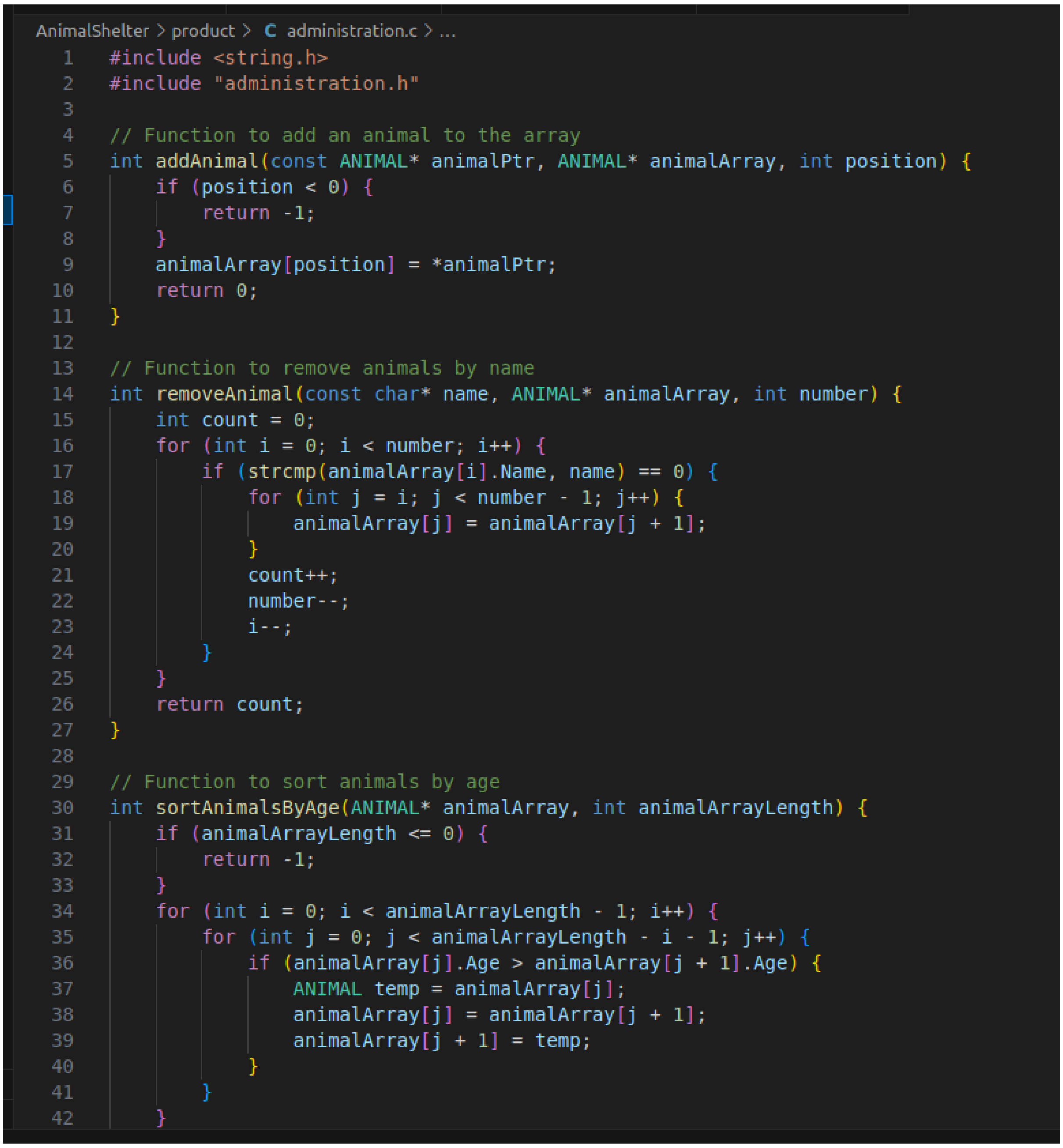
```c
void findAnimalByName() {
      // Code to find animal by name
}
```

5. main()
  - Purpose: Main entry point of the program.
  - Parameters: None

- Returns: int
- Description: Displays the menu and handles user input.

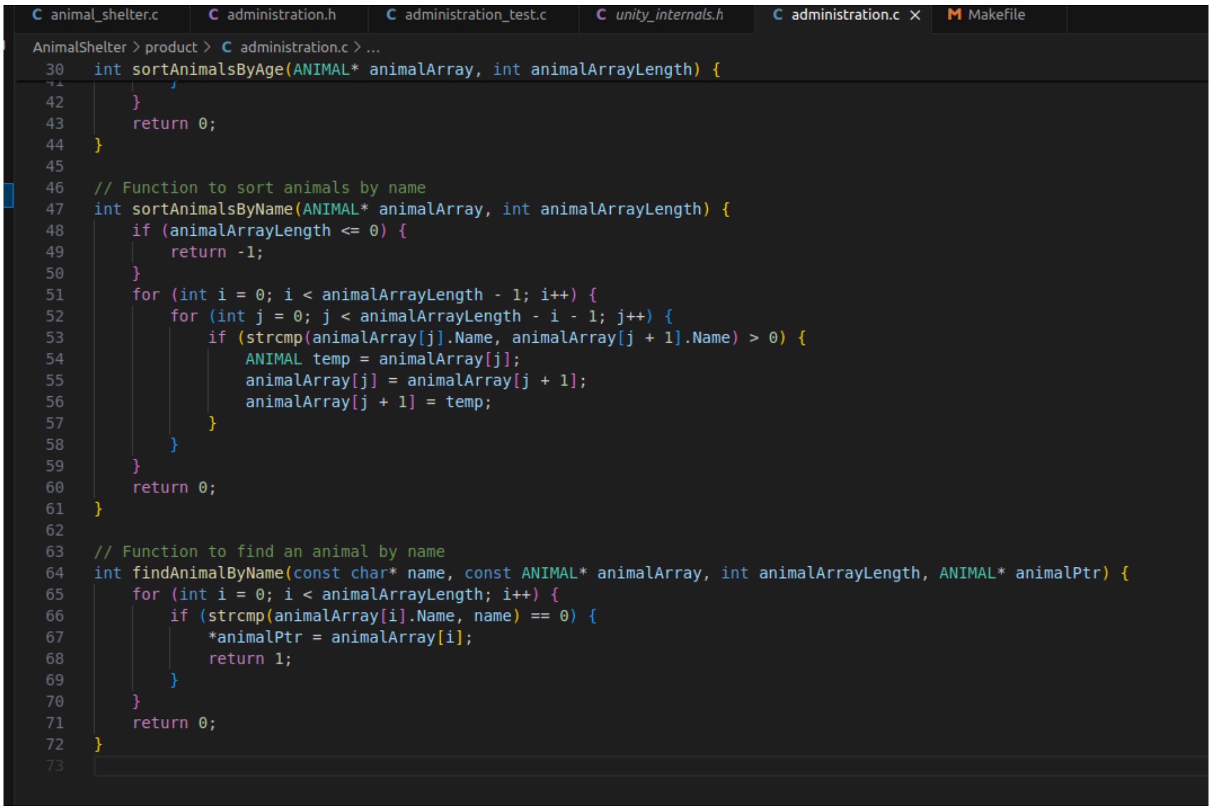
```c
int main() {
	// Main program loop
}
```

Code For administration.c:



AnimalShelter > product > C administration.c > ...

```c
#include <string.h>
#include "administration.h"

// Function to add an animal to the array
int addAnimal(const ANIMAL* animalPtr, ANIMAL* animalArray, int position) {
    if (position < 0) {
        return -1;
    }
    animalArray[position] = *animalPtr;
    return 0;
}

// Function to remove animals by name
int removeAnimal(const char* name, ANIMAL* animalArray, int number) {
    int count = 0;
    for (int i = 0; i < number; i++) {
        if (strcmp(animalArray[i].Name, name) == 0) {
            for (int j = i; j < number - 1; j++) {
                animalArray[j] = animalArray[j + 1];
            }
            count++;
            number--;
            i--;
        }
    }
    return count;
}

// Function to sort animals by age
int sortAnimalsByAge(ANIMAL* animalArray, int animalArrayLength) {
    if (animalArrayLength <= 0) {
        return -1;
    }
    for (int i = 0; i < animalArrayLength - 1; i++) {
        for (int j = 0; j < animalArrayLength - i - 1; j++) {
            if (animalArray[j].Age > animalArray[j + 1].Age) {
                ANIMAL temp = animalArray[j];
                animalArray[j] = animalArray[j + 1];
                animalArray[j + 1] = temp;
            }
        }
    }
```



```c
int sortAnimalsByAge(ANIMAL* animalArray, int animalArrayLength) {
        }
    return 0;
}

// Function to sort animals by name
int sortAnimalsByName(ANIMAL* animalArray, int animalArrayLength) {
    if (animalArrayLength <= 0) {
        return -1;
    }
    for (int i = 0; i < animalArrayLength - 1; i++) {
        for (int j = 0; j < animalArrayLength - i - 1; j++) {
            if (strcmp(animalArray[j].Name, animalArray[j + 1].Name) > 0) {
                ANIMAL temp = animalArray[j];
                animalArray[j] = animalArray[j + 1];
                animalArray[j + 1] = temp;
            }
        }
    }
    return 0;
}

// Function to find an animal by name
int findAnimalByName(const char* name, const ANIMAL* animalArray, int animalArrayLength, ANIMAL* animalPtr) {
    for (int i = 0; i < animalArrayLength; i++) {
        if (strcmp(animalArray[i].Name, name) == 0) {
            *animalPtr = animalArray[i];
            return 1;
        }
    }
    return 0;
}
```

Purpose: Implements the backend functions for animal data manipulation.

Key Functions:

1. addAnimal(const ANIMAL* animalPtr, ANIMAL* animalArray, int position)

- Purpose: Adds an animal to the array.

- Parameters: `animalPtr` (pointer to the animal to add), `animalArray` (array of animals), `position` (index to add the animal).

- Returns: int (status code)

```c
int addAnimal(const ANIMAL* animalPtr, ANIMAL* animalArray, int position) {
    // Code to add animal
}
```

2. removeAnimal(const char* name, ANIMAL* animalArray, int number)

- Purpose: Removes animals by name.

  - Parameters: `name` (name of the animal to remove), `animalArray` (array of animals), `number` (number of animals).
  - Returns: int (status code)

```c
int removeAnimal(const char* name, ANIMAL* animalArray, int number) {
	// Code to remove animal
}
```

3**. sortAnimalsByAge(ANIMAL* animalArray, int animalArrayLength)**
  - Purpose: Sorts animals by age.
  - Parameters: `animalArray` (array of animals), `animalArrayLength` (length of the array).
  - Returns: int (status code)

```c
int sortAnimalsByAge(ANIMAL* animalArray, int animalArrayLength) {
	// Code to sort animals by age
}
```

4. sortAnimalsByName(ANIMAL* animalArray, int animalArrayLength)
  - Purpose: Sorts animals by name.
  -Parameters: `animalArray` (array of animals), `animalArrayLength` (length of the array).
  -Returns: int (status code)

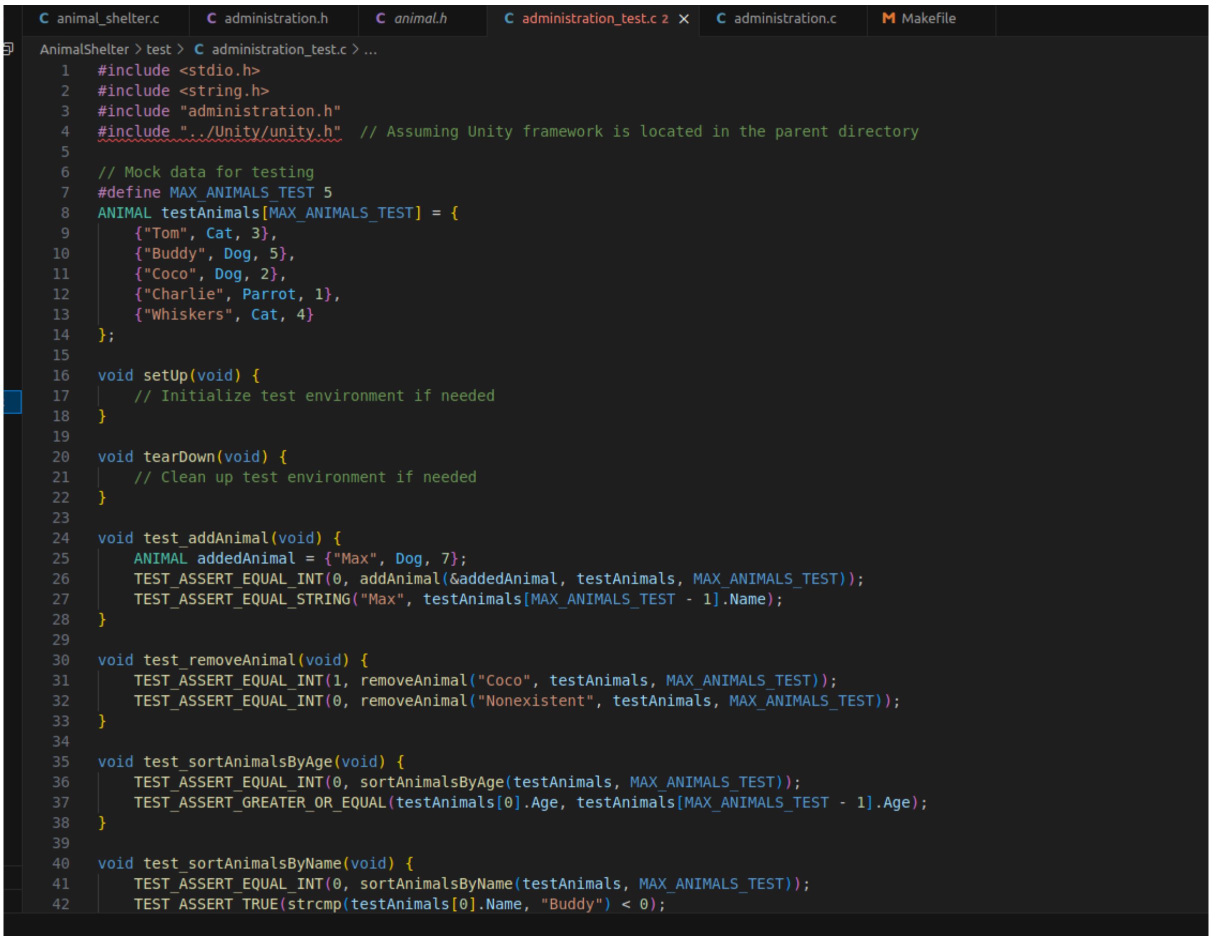
```c
int sortAnimalsByName(ANIMAL* animalArray, int animalArrayLength) {
	// Code to sort animals by name
}
```

5. findAnimalByName(const char* name, const ANIMAL* animalArray, int animalArrayLength, ANIMAL* animalPtr)
  - Purpose: Finds an animal by name.
  - Parameters: `name` (name of the animal), `animalArray` (array of animals), `animalArrayLength` (length of the array), `animalPtr` (pointer to store the found animal).

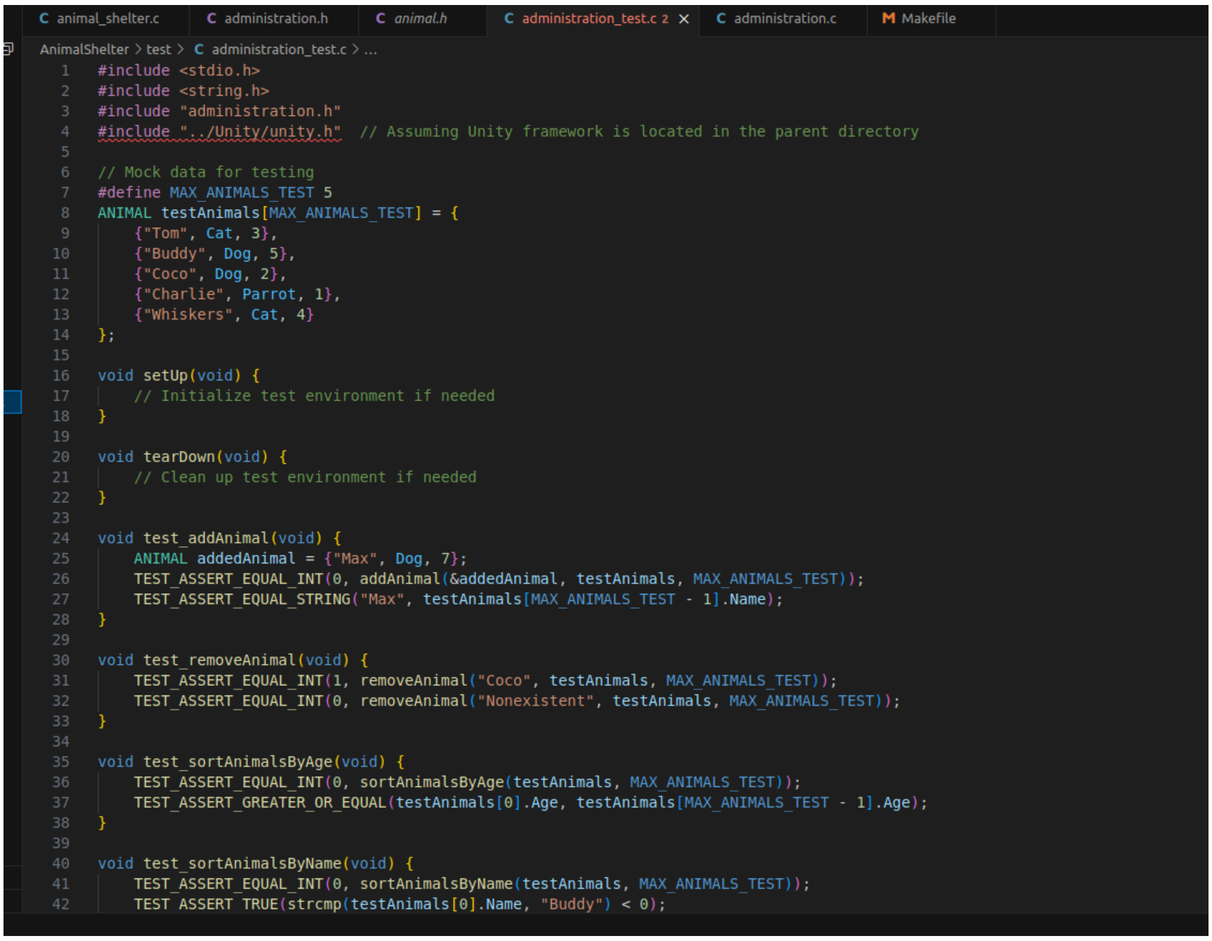
- Returns: int (status code)

```c
int findAnimalByName(const char* name, const ANIMAL* animalArray, int animalArrayLength, ANIMAL* animalPtr) {
        // Code to find animal by name
}
```

Code for: administration_test.c



```c
#include <stdio.h>
#include <string.h>
#include "administration.h"
#include "../Unity/unity.h"  // Assuming Unity framework is located in the parent directory

// Mock data for testing
#define MAX_ANIMALS_TEST 5
ANIMAL testAnimals[MAX_ANIMALS_TEST] = {
    {"Tom", Cat, 3},
    {"Buddy", Dog, 5},
    {"Coco", Dog, 2},
    {"Charlie", Parrot, 1},
    {"Whiskers", Cat, 4}
};

void setUp(void) {
    // Initialize test environment if needed
}

void tearDown(void) {
    // Clean up test environment if needed
}

void test_addAnimal(void) {
    ANIMAL addedAnimal = {"Max", Dog, 7};
    TEST_ASSERT_EQUAL_INT(0, addAnimal(&addedAnimal, testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_EQUAL_STRING("Max", testAnimals[MAX_ANIMALS_TEST - 1].Name);
}

void test_removeAnimal(void) {
    TEST_ASSERT_EQUAL_INT(1, removeAnimal("Coco", testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_EQUAL_INT(0, removeAnimal("Nonexistent", testAnimals, MAX_ANIMALS_TEST));
}

void test_sortAnimalsByAge(void) {
    TEST_ASSERT_EQUAL_INT(0, sortAnimalsByAge(testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_GREATER_OR_EQUAL(testAnimals[0].Age, testAnimals[MAX_ANIMALS_TEST - 1].Age);
}

void test_sortAnimalsByName(void) {
    TEST_ASSERT_EQUAL_INT(0, sortAnimalsByName(testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_TRUE(strcmp(testAnimals[0].Name, "Buddy") < 0);
```



```c
#include <stdio.h>
#include <string.h>
#include "administration.h"
#include "../Unity/unity.h"  // Assuming Unity framework is located in the parent directory

// Mock data for testing
#define MAX_ANIMALS_TEST 5
ANIMAL testAnimals[MAX_ANIMALS_TEST] = {
    {"Tom", Cat, 3},
    {"Buddy", Dog, 5},
    {"Coco", Dog, 2},
    {"Charlie", Parrot, 1},
    {"Whiskers", Cat, 4}
};

void setUp(void) {
    // Initialize test environment if needed
}

void tearDown(void) {
    // Clean up test environment if needed
}

void test_addAnimal(void) {
    ANIMAL addedAnimal = {"Max", Dog, 7};
    TEST_ASSERT_EQUAL_INT(0, addAnimal(&addedAnimal, testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_EQUAL_STRING("Max", testAnimals[MAX_ANIMALS_TEST - 1].Name);
}

void test_removeAnimal(void) {
    TEST_ASSERT_EQUAL_INT(1, removeAnimal("Coco", testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_EQUAL_INT(0, removeAnimal("Nonexistent", testAnimals, MAX_ANIMALS_TEST));
}

void test_sortAnimalsByAge(void) {
    TEST_ASSERT_EQUAL_INT(0, sortAnimalsByAge(testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_GREATER_OR_EQUAL(testAnimals[0].Age, testAnimals[MAX_ANIMALS_TEST - 1].Age);
}

void test_sortAnimalsByName(void) {
    TEST_ASSERT_EQUAL_INT(0, sortAnimalsByName(testAnimals, MAX_ANIMALS_TEST));
    TEST_ASSERT_TRUE(strcmp(testAnimals[0].Name, "Buddy") < 0);
```

Purpose: Contains unit tests for `administration.c` functions using the Unity framework.

Key Tests:

1. **test_addAnimal()**
   - Purpose: Tests the `addAnimal` function.
   - **Parameters:** None
   - **Returns:** None
   - **Description:** Adds a mock animal to the array and verifies it was added correctly.

```c
void test_addAnimal() {
    // Code to test addAnimal
}
```

2**. test_removeAnimal()**
- **Purpose:** Tests the `removeAnimal` function.
- **Parameters:** None
- **Returns:** None
- **Description:** Removes a mock animal from the array and verifies it was removed correctly.

```c
void test_removeAnimal() {
    // Code to test removeAnimal
}
```

3. **test_sortAnimalsByAge()**
- **Purpose:** Tests the `sortAnimalsByAge` function.
- **Parameters:** None
- **Returns:** None
- **Description:** Sorts mock animals by age and verifies the order is correct.

```c
void test_sortAnimalsByAge() {
    // Code to test sortAnimalsByAge
}
```

4. **test_sortAnimalsByName()**
- **Purpose:** Tests the `sortAnimalsByName` function.
- **Parameters:** None
- **Returns:** None
- **Description**: Sorts mock animals by name and verifies the order is correct.

```c
void test_sortAnimalsByName() {
    // Code to test sortAnimalsByName
}
```

5. **test_findAnimalByName()**
- **Purpose:** Tests the `findAnimalByName` function.
- **Parameters:** None

- Returns: None
- Description: Finds a mock animal by name and verifies the correct animal is found.

```c
void test_findAnimalByName() {
      // Code to test findAnimalByName
}
```

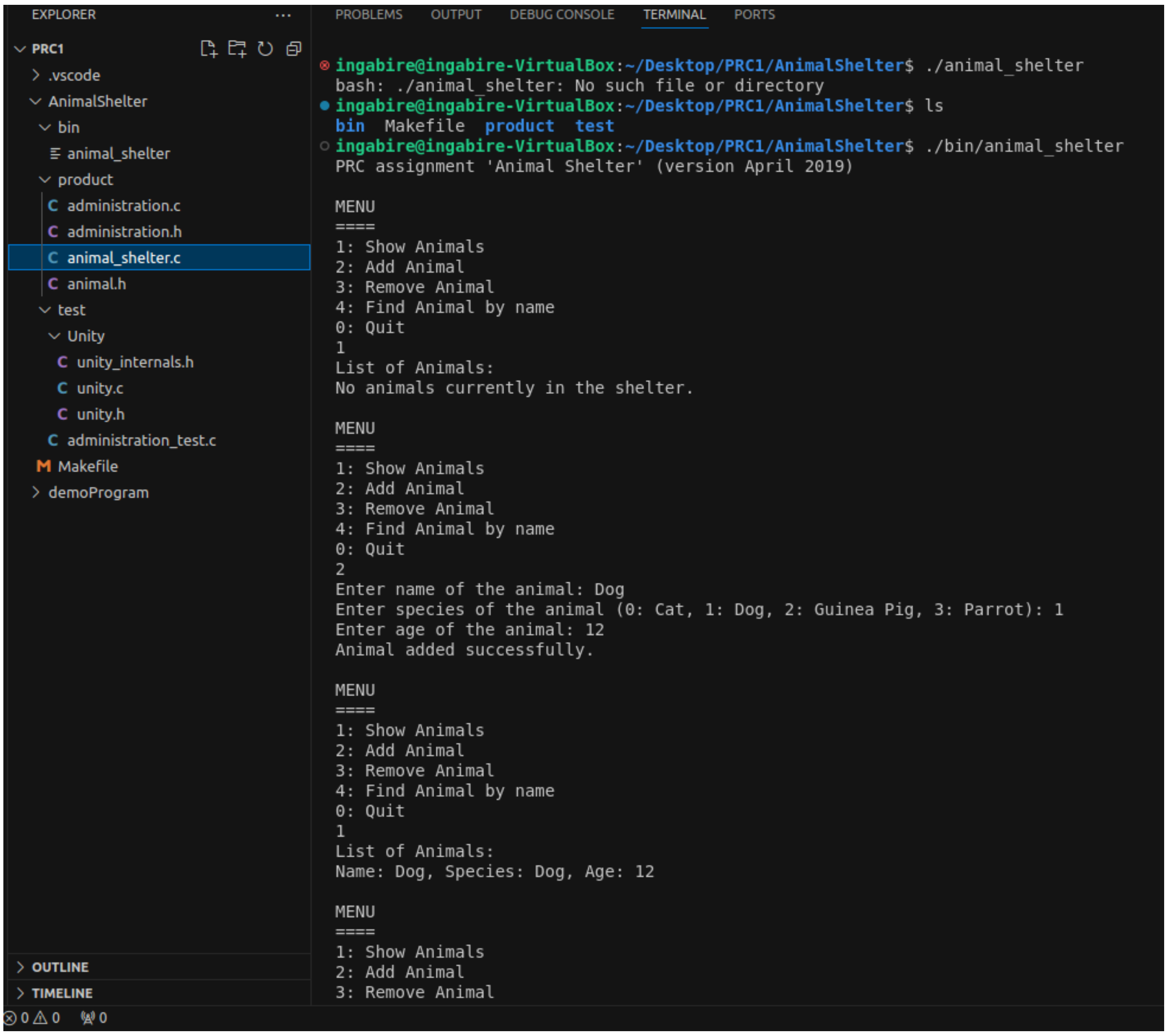
Build and Run Instructions

Build the Project
1. Open a terminal.
2. Navigate to the project directory.
3. Run the following command to compile the project:
```sh
make
```

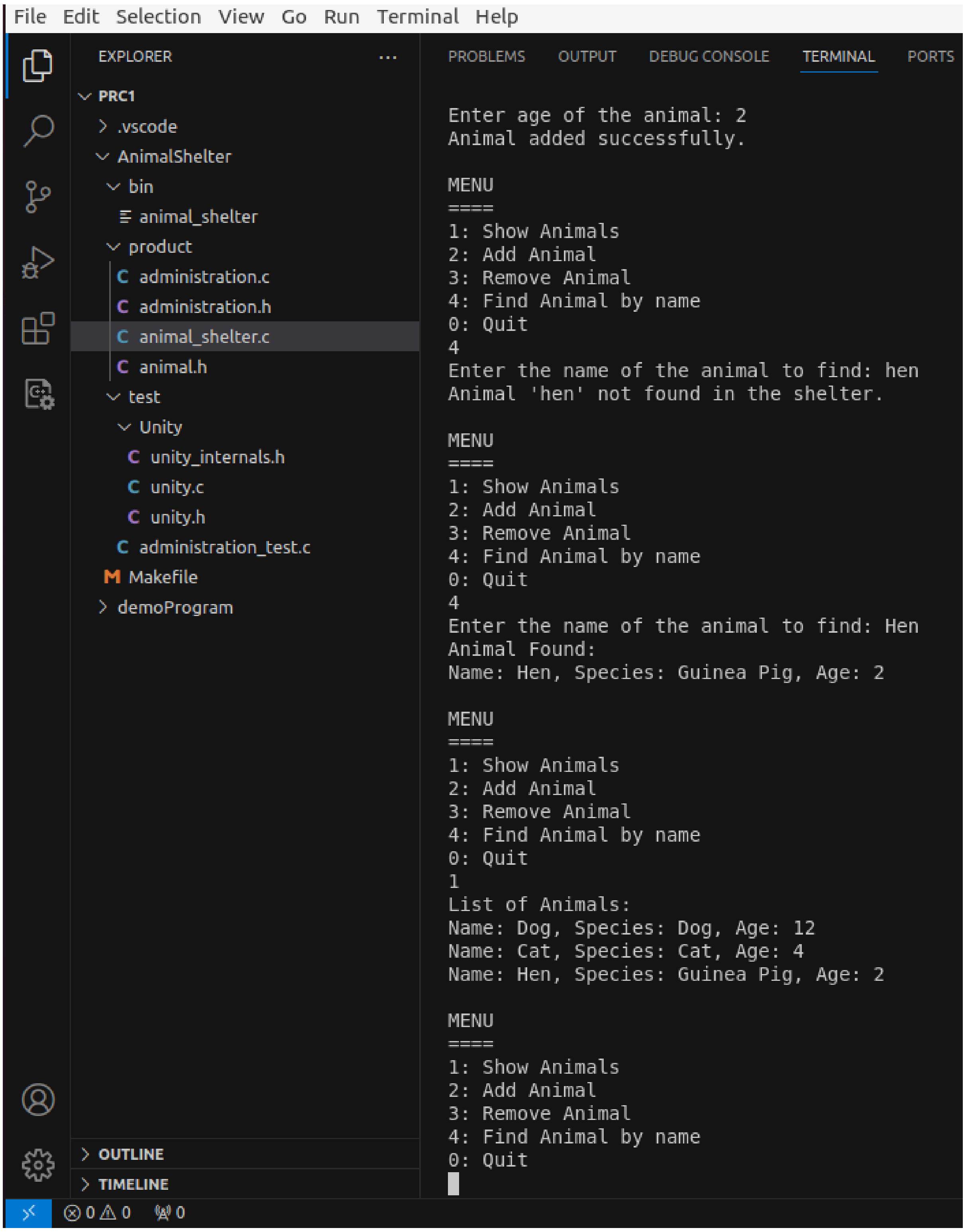
Run the Program:
1. In the terminal, run the executable:
```sh
./bin/animal_shelter
```

Run the Tests:
1. In the terminal, run the test executable:
```sh
make adminTest
./bin/administrationTest
```

- **Test Results**: Screenshot of the test results output.



```
ingabire@ingabire-VirtualBox:~/Desktop/PRC1/AnimalShelter$ ./animal_shelter
bash: ./animal_shelter: No such file or directory
ingabire@ingabire-VirtualBox:~/Desktop/PRC1/AnimalShelter$ ls
bin  Makefile  product  test
ingabire@ingabire-VirtualBox:~/Desktop/PRC1/AnimalShelter$ ./bin/animal_shelter
PRC assignment 'Animal Shelter' (version April 2019)

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
1
List of Animals:
No animals currently in the shelter.

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
2
Enter name of the animal: Dog
Enter species of the animal (0: Cat, 1: Dog, 2: Guinea Pig, 3: Parrot): 1
Enter age of the animal: 12
Animal added successfully.

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
1
List of Animals:
Name: Dog, Species: Dog, Age: 12

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
```



```
Enter age of the animal: 2
Animal added successfully.

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
4
Enter the name of the animal to find: hen
Animal 'hen' not found in the shelter.

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
4
Enter the name of the animal to find: Hen
Animal Found:
Name: Hen, Species: Guinea Pig, Age: 2

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
1
List of Animals:
Name: Dog, Species: Dog, Age: 12
Name: Cat, Species: Cat, Age: 4
Name: Hen, Species: Guinea Pig, Age: 2

MENU
====
1: Show Animals
2: Add Animal
3: Remove Animal
4: Find Animal by name
0: Quit
```

Conclusion

Summary: The Animal Shelter Management System successfully manages the animal data, allowing for adding, removing, sorting, and finding animals. The system's functionality is verified through comprehensive unit tests.

Future Improvements: Potential future enhancements could include a graphical user interface, a database backend for persistent storage, and additional features such as animal adoption management.