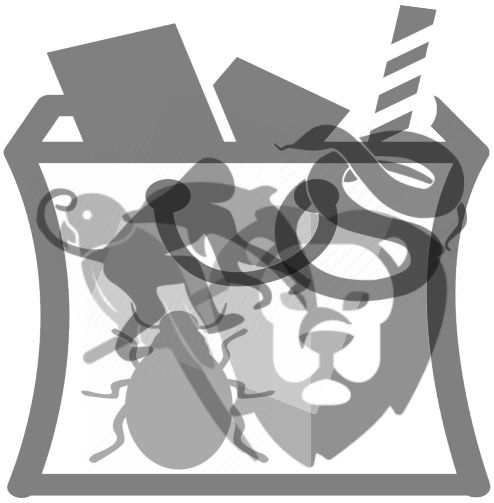
Project 3: ZooRecord "is-a" Bag of Animals



Project3 will continue to build on Projects 1 and 2 to work with **Inheritance, Templates, ADTs and file input.** After completion of this project you must be very comfortable with compiling multiple classes as well as template classes (i.e. you do not compile them!!!) into one program, instantiating objects of a template class, basic inheritance and reading input from a file. If you are not absolutely comfortable with all of this, please seek help immediately (contact me or visit the lab to work with our UTAs).

In Project3 we will work with the `ArrayBag` class discussed during lecture. This project consists of **two parts**:

1. Modify the `ArrayBag` class
2. Implement a class `ZooRecord` which inherits from `ArrayBag` and stores `Animal` objects (you will also make a small modification to the `Animal` class to support `ZooRecord`)

You will find the `ArrayBag` class (as discussed in lecture) on Blackboard under Course Materials / Project3.

First you **must read the `ArrayBag` interface** and understand how it works. You will need to know how to use `ArrayBag` objects by understanding its interface.

Note: Reading interfaces is the way you learn to use language libraries, so this is great practice for that!

Implementation - 2 parts:

Work incrementally! Start from Part 1 (implement and test), when that runs correctly then move on to Part2.

Part1- ArrayBag modifications:

1. Modify the `add` method so that it will not allow duplicate items to be added to the ArrayBag (conceptually the Bag becomes a Set).

2. Implement public method `display()` to display the contents of the bag to standard output in the form `"item1, item2, … , itemN\n"`

```
/**@post prints the contents of items_ to the standard output
       separated by commas and followed by a new line.**/
void display() const;
```

3. Overload public `operator+=` to implement Set Union (**Hint:** you can use other ArrayBag operations to implement this)

```
/** implements Set Union
 The union of two sets A and B is the set of elements which are in A,
 in B, or in both A and B.
@param a_bag to be combined with the contents of this (the calling) bag
@post adds as many items from a_bag as space allows
*/
void operator+=(const ArrayBag<T>& a_bag);
```

Note: Because ArrayBag is of fixed size, += will only copy as many items from a_bag as there is space available without deleting its original contents. **Notice how fixed size can be an issue and force unintuitive implementations**! We will address the fixed-size problem soon.

4. Overload public `operator-=` to implement Set Difference (**Hint:** you can use other ArrayBag operations to implement this)

```
/** implements Set Difference
 The (set) difference between two sets A and B is the set that
 consists of the elements of A which are not elements of  B
 @param a_bag to be subtracted from this (the calling) bag
 @post removes all data from items_ that is also found in a_bag
*/
void operator-=(const ArrayBag<T>& a_bag);
```

5. Overload public `operator/=` to implement Set Intersection (**Hint:** you can use other ArrayBag operations to implement this)

```
/** implements Set Intersection
 The intersection of two sets A and B is the set that
 consists of the elements that are in both A and B
 @param a_bag to be intersected with this (the calling) bag
 @post items_ no longer contains data not found in a_bag
*/
void operator /=(const ArrayBag<T>& a_bag);
```

Note that the overloaded operators are **+=** , **−=** and **/=** (not +, – and /). Note also that these are all **`void`** functions. This means that these do not produce a new array that contains the union, difference and intersection respectively. Instead, they modify `items_` to contain the union, difference and intersection respectively. Thus, in our set analogy, `items_` corresponds to both set A and the resulting union/difference/intersection, while `a_bag` corresponds to set B.

IMPORTANT: Please remember that you DO NOT compile (or include in your project) the implementation (.cpp) of a Template class. Please look at slide 38 from Lecture 3 and make sure you understand separate compilation with templates (it will probably help if, as suggested on slide 40, you first run a dummy test and make sure you can compile a simple/trivial template class)

Part2 - ZooRecord class:

Write a class, `ZooRecord`, that **inherits from `ArrayBag`** and stores `Animal` objects. The `ZooRecord` class should have at least the following public methods:

1. `ZooRecord(); //default constructor for empty record`

2.
```
/**parameterized constructor
 @pre the input file is expected to be in CSV
 (comma separated value) format as:
 "animal_name,hair,feathers,eggs,milk,airborne,aquatic,predator,toothed,
 backbone,breathes,venomous,fins,legs,tail,domestic,catsize,class_type\n"
 @param input_file_name the name of  the input file
 @post adds Animal objects to record as per the data in the input file
 **/
ZooRecord(std::string input_file_name);
```

3.
```
/**@post displays all animals in record, one per line by calling animal's
display method" **/
 void display();
```

You must also modify the **Animal class** as follows:

4. Add the `display()` method:

```
/**@post displays animal data in the form:
  "animal_name is [not] domestic and [it is / is not] a predator\n"
  **/
 void display();
```

5. Overload `operator==` for the `Animal` class, otherwise you will have a problem when the `add()` methods tries to compare two animals to add them to the `ZooRecord`. This operator can be a friend function, since it does not "belong" to either of the two Animals being compared,. *This statement is not trivial, if you don't understand please ask for help from the UTAs in lab!*

Testing:

Part1 - Testing your modification to ArrayBag:

Before you move to Part2, YOU MUST make sure that your modifications to ArrayBag work correctly. To do so write your own main function (not for submission) that does the following:

- Instantiate two ArrayBag objects that stores integers
- Add integers to the two bags (some integers should be common to the two bags)
- Call += on one of the bags, display its contents and make sure the operation worked correctly (i.e. bag1 ∪ bag2) and that your modification to add worked s.t. there are no duplicates. Also **be sure your `dispaly()` function displays the contents of the bag as specified** in Part1 above (Gradescope will rely on `display()` to check that other functions work correctly)
- Call −= on one of the bags, display its contents and make sure the operation worked correctly (i.e. bag1 − bag2).
- Repopulate the bags s.t. some of their contents overlap, then call /= on one of the bags, display its contents and make sure the operation worked correctly (i.e. bag1 ∩ bag2).

Part2 - Testing the ZooRecord class:

Again in a main function (not for submission) do the following

- Instantiate a ZooRecord object with the name of an input file
- Display the record and make sure that all animals in the file were added to the ZooRecord. Again **be sure your `display()` function here has been overloaded to display Animals EXACTLY as specified above** in Part2 above.

The Data:

The input file will be in **csv** (comma separated value) format, and each line corresponds to one `Animal`.
The data comes from the UCI Machine Learning repository (https://archive.ics.uci.edu/ml/datasets/zoo) , it consists of 101 animals from a zoo. There are 16 variables with various traits to describe the animals. The 7 Class Types are: Mammal (1), Bird (2), Reptile (3), Fish (4), Amphibian (5), Bug (6) and Invertebrate (7)
This dataset is often used to predict the classification of the animals based upon the variables using machine learning techniques. In this course we will use it to illustrate the concept of Inheritance in OOP.
Each line in the input file has the following format:

```
animal_name,hair,feathers,eggs,milk,airborne,aquatic,predator,toothed,backbone,breathes,venomous,fins,legs,tail,domestic,catsize,class_type
```

For this project you will only be concerned with those attributes found in the `Animal` class from Project 2, namely `animal_name`, `predator` and `domestic`. For each line read from the input file, the `ZooRecord` parameterized constructor will create an `Animal` object with these attributes and add it to the record.
You can find the input file named `zoo.csv` on Blackboard under Course Materials / Project3

Review – reading the input (REVIEW):

In C++ to read input from a file you need a file stream

```
#include <fstream>
```

Since we are only reading input you can use an `ifstream` object.
Since we are reading from a csv file, every animal is on a line. On each line, each data item is separated by a comma.
You may use `string::getline()` to read lines from the `ifstream`.

```
#include <string>
```

You may find it useful to use a stringstream to then read each piece of data (id, first_name and last_name) from each line you read from the input file.

```
#include <sstream>
```

You may use `getline()` to read data from the `sstream` as well. Remember that `getline()` may take a delimiter. The default delimiter is `'\n'`, but if you are reading comma-separated values you can use `','` as the delimiter.

```
getline(stream, variable, delimiter);
```

Don't forget to:
- Open the stream before reading.
- Check that opening the stream did not fail before reading, and output (`cout`) an error message if it does fail.
- Close the stream after reading.

Note: **Reading from input file should be familiar from CSci 135**. If you need to review, lookup `ifstream, sstream` and `string::getline()`
References for each of these are easily found online:
http://www.cplusplus.com/reference/fstream/ifstream/
http://www.cplusplus.com/reference/sstream/stringstream/
http://www.cplusplus.com/reference/string/string/getline/
If you need help with this even after having reviewed the documentation, please don't hesitate to ask for help from the tutors in labB or make an appointment with me. After this project you will be expected to be able to read from csv files.

Grading Rubric:

- **Correctness 80%** (distributed across unit testing of your submission)
- **Documentation 10%**
- **Style and Design 10%** (proper naming, modularity and organization)
- A submission that implements all required classes and/or functions but does not compile will receive 40 points total (including documentation and design).

Submission:

For this project you will submit **6 files: `ArrayBag.hpp, ArrayBag.cpp, ZooRecord.hpp, ZooRecord.cpp, Animal.hpp, Animal.cpp`**
You must submit all files. If you only complete Part 1, for partial credit you must also submit the ZooRecord files, even if only a "dummy" or empty class. The idea of a dummy class is analogous to that of a dummy function.If you need further clarification on how to create a dummy class please ask the UTAs in lab.

Your project must be submitted on Gradescope.

Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You MUST test and debug your program locally.
Before submitting to Gradescope you MUST ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope.
"But it ran on my machine!" is not a valid argument for a submission that does not compile.
Once you have done all the above you submit it to Gradescope.

The due date is Friday September 27 by 5pm. No late submissions will be accepted.

Have Fun!!!

Image credits: https://www.iconfinder.com