

Assignment #2

Adapted from Professor Nico's Assignment
Decomposition Outline Due: January 27, 2017 11:59pm
Program Due: February 1, 2017 11:59pm

Overview

For this assignment you will implement a guessing game that tries to guess what the user is thinking by asking a series of questions and that “learns” from incorrect guesses. The “learning” takes place when a guess is incorrect; the program will prompt the user for an additional question that would have led to a correct guess.

Topics: dynamic memory management, files, simple text processing

Decomposition Outline Document

You must submit a program decomposition outline by the posted due date. You can use the simple example given with the first assignment as a model, but keep in mind that this program is a little more involved. Be certain to include enough detail to make it clear that you have given careful thought to the construction of this program. In addition, you should address your intended file format by specifying how it will look with two questions and the corresponding answers.

Details

Though this program could be used to implement a guessing game for any topic, the narrative below assumes that the goal is to guess animals.

The program, called `guess`, uses a database of questions stored in a file called `qa.db`¹. The general structure of this database is a decision tree with questions at the interior nodes and answers (animals) at the leaf nodes. `guess` starts at the root of the tree, asking yes/no questions until it reaches an animal. When it reaches an animal, it asks “Is it ...?”. If the program's guess is correct, it prints a victory message and thanks the user. If the program's guess is incorrect, however, it learns from its mistakes. First, it admits its ignorance, then it asks for:

1. the correct answer with the appropriate article (e.g., “a duck” rather than just “duck”),
2. a yes/no question that can differentiate the correct animal from the final guess that failed, and
3. the correct answer for the new animal

With this information it updates its decision tree and writes it back to `qa.db` so that it will be prepared with the new question the next time it is run.

Specifics

- Your program may not put any arbitrary limit on the length of strings (animal names or questions). When it reads a string it must get everything up to the newline.
- The only character you may assume is not part of either a question or an animal's name is the newline character.
- Your program should interpret any response starting with 'y' or 'Y' as “yes” and anything else as “no”. That is, “yRgflrts” is an affirmative response, while “right on!” is a negative one.
- The only exception to the above rule is an empty string. An empty string is neither affirmative nor negative and requires the program to re-ask. (Note that an empty string in this context means that nothing was read before the newline character.)
- Reading a yes/no response should consume the entire line of input up to and including the newline (to prevent buffered characters from being construed as the answer to the following question).

¹This is a database only in the simplest sense: it stores data. The file format, discussed more later, is entirely up to you.

- If `qa.db` doesn't exist when your program is run, your program should create it. The program will, of course, be forced to give up immediately since it doesn't know about any animals (since the database is empty).
- Your program is not expected to “clean” the user input. This means that if the user gives nonsensical responses, then the program will repeat those responses as-is.
- In order to ensure consistent behavior among submissions:
 - Your database file *must* be called `qa.db`
 - Other than the questions provided by the player in the database, the program's dialog should be the strings shown in Table 1 and in the examples.
 - Each run of the program must consist of exactly one round of the game and no more. That is, don't add a “do you want to play again?” query, or other extraneous input. The only IO should be the questions and answers specified above.

Occasion	String
Creating a new database	Unable to read <database name>. Starting fresh.
Empty database (when creating the tree)	What is it (with article)?
Guessing	Is it <animal name>
After a loss (extending the tree)	How disappointing. What is it (with article)?
After the name	What is a yes/no question that will distinguish <newanimal> from <oldanimal>? ?
After the question	What is the answer to the question for <newanimal>?
After the response	I'll get it next time, I'm sure. Thanks for playing.
After a win	My, am I clever. :) Thanks for playing.

Table 1: Strings used in `guess`

Example

Figure 2 shows the outcome of a series of `guess` runs assuming that `guess`'s initial database looks like the one pictured as in Figure 1. The first run of the program represents a winning run. The second run is a loss, but this loss leads to the modification of the decision tree shown, so that it wins on the third attempt.

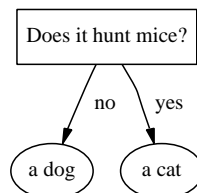


Figure 1: Initial state of `qa.db`

Tips

The overall approach and decomposition are to be determined by you. The following are some tips to get you thinking.

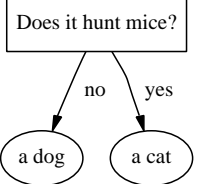
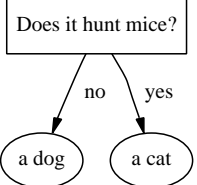
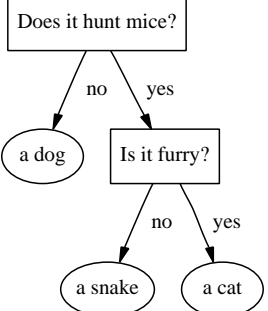
Run	Transcript	DecisionTree
1	<pre> % ./guess Does it hunt mice? no Is it a dog? yes My, am I clever. :) Thanks for playing. %</pre>	 <pre> graph TD A[Does it hunt mice?] -- no --> B((a dog)) A -- yes --> C((a cat)) </pre>
2	<pre> % ./guess Does it hunt mice? yes Is it a cat? no How disappointing. What is it (with article)? a snake What is a yes/no question that will distinguish a snake from a cat? ? Is it furry? What is the answer to the question for a snake? no I'll get it next time, I'm sure. Thanks for playing. %</pre>	 <pre> graph TD A[Does it hunt mice?] -- no --> B((a dog)) A -- yes --> C((a cat)) </pre>
3	<pre> % ./guess Does it hunt mice? yes Is it furry? no Is it a snake? yes My, am I clever. :) Thanks for playing. %</pre>	 <pre> graph TD A[Does it hunt mice?] -- no --> B((a dog)) A -- yes --> C[Is it furry?] C -- no --> D((a snake)) C -- yes --> E((a cat)) </pre>

Figure 2: Example: modification of qa.db

1. Give considerable thought to how you are going to format your `qa.db` file. A good choice will make it easy to store your decision tree and reconstruct it later. A poor choice will make this difficult.

When designing your format, keep in mind that the things you can assume about question or answer strings is that they will not be empty and that they will not contain the newline character.

(For what it's worth, the format of my datafile is a preorder traversal of the tree with strings delimited by newlines and null children represented as empty strings.)

2. The format of your `qa.db` file is for you to decide and, as such, how your program will detect a corrupted file (e.g., somebody deletes a line from the file) is for you to determine. Your program should handle such errors gracefully by either trying to recover or by reporting the issue and exiting.
3. Keep things simple. This program does not require terribly complex structures. If you find yourself building something very complicated or making a lot of exceptions, you may want to step back and rethink your approach.
4. Be *very* careful with memory management. Be sure to `free()` everything you allocate when you're done with it, but only `free()` things you have allocated. Similarly, be sure only to `fclose()` files you have successfully `fopen()`ed.

valgrind

For full credit, your submission must cleanly pass a `valgrind` memory check on the department servers (e.g., `unix13.csc.calpoly.edu`). Refer back to the `valgrind` quick start for directions on running the memory check.

Submit

- All source code.
- A `Makefile` that can be used to build your program (with the resulting executable named `guess`).

Grading

The ability to test some functionality requires that other parts work correctly (e.g., I cannot test that your program properly extends the database if the file manipulation is not working, nor can I test the file manipulation if the tree cannot be extended). As such, do not use the grading breakdown as a decomposition or implementation guide.

Feature	Percentage
Decomposition Document	10
Reasonable Final Decomposition	5
Core Functionality	60
Error Handling	10
— e.g., missing file, corrupted file, empty string input	
Valgrind Check	15

Sample Runs

The following is a sample series of runs to demonstrate the learning behavior. When it starts, `qa.db` does not exist.

I have also installed an executable version of `guess` on the department servers in `~akeen/bin/guess` so you can run it yourself.

```
% ls qa.db
ls: cannot access qa.db: No such file or directory
% ./guess
qa.db: No such file or directory
Unable to read database qa.db. Starting fresh.
```

What is it (with article)? an elephant

```
% ./guess
Is it an elephant? yes
```

My, am I clever. :)
Thanks for playing.

```
% ./guess
Is it an elephant? no
```

How disappointing.

What is it (with article)? a duck

What is a yes/no question that will distinguish a duck
from an elephant?
? Can it swim?

What is the answer to the question for a duck? yes

I'll get it next time, I'm sure.
Thanks for playing.

```
% ./guess
Can it swim? yes
Is it a duck? no
```

How disappointing.

What is it (with article)? a giant squid

What is a yes/no question that will distinguish a giant squid
from a duck?
? Does it have feathers?

What is the answer to the question for a giant squid? no

I'll get it next time, I'm sure.
Thanks for playing.

```
% ./guess
Can it swim? no
Is it an elephant? no
```

How disappointing.

What is it (with article)? a worm

What is a yes/no question that will distinguish a worm
from an elephant?

? Is it smaller than a breadbox?

What is the answer to the question for a worm? yes

I'll get it next time, I'm sure.

Thanks for playing.

% ./guess

Can it swim? no

Is it smaller than a breadbox? yes

Is it a worm? yes

My, am I clever. :)

Thanks for playing.

% ls -l qa.db

-rw-----. 1 akeen domain^users 115 Jan 22 14:45 qa.db

%