Lab 6

Due Dec 1, 2017

November 15, 2017

1 Description

Over the course of the next few labs, you and a partner will develop an interpreter for interactive fiction. Interactive fiction (IF) is a text-based form of entertainment in which a reader is given passages of text interspersed with choices. In a sense, these resemble the "Choose Your Own Adventures" that were once popular; however, IF may be more sophisticated in that early choices in the story may influence later passages or choices.

In this lab, you and your partner will submit your IF interpreter code, as well as submitting a peer evaluation.

2 Submission instructions and grading

For this lab, you should submit two items. To the group assignment, submit a zip file containing your source (.cpp and .h) files, and to the individual assignment, submit your peer evaluation.

Your code will be compiled using `g++`, so you should make sure that your code can be compiled with `g++` before submitting. Many of the test cases will likely be drawn from the Lab 5 submissions, so you are encouraged to make sure that your code can successfully pass your test cases.

3 Peer evaluation

Your peer evaluations should indicate four things:

1. All group members' names
2. For each group member, what part of the code did they implement
3. For each group member, a numeric score indicating their level of participation in the project (Labs 4–6)
    - These scores should sum up to 10 (or 15 for a group of three)
4. Any notes, positive or negative, that you think the grader should know when evaluating your group members' teamwork contribution

A Specifications

Your IF interpreter will be based on a limited subset of Harlowe (`https://twinery.org/wiki/harlowe:reference`), a common format for IF.

A.1 Passages

Interactive fiction works are divided into *passages*, which appear in the HTML tags `<tw-passagedata>`. Each passage will start with `<tw-passagedata ...>` and will end with `</tw-passagedata>`. In addition to starting with `<tw-passagedata`, the opening tag will specify 4 attributes, `pid`, `name`, `tags`, and `location`, and the body of the passage will be between the opening and closing tags.

Example passage:

```
<tw-passagedata pid="1" name="start" tags="" location="100,100">
The body of the passage will be here.
</tw-passagedata>
```

Your interpreter only needs to pay attention to the `name` of each passage, the `pid`, `tags`, and `location` can be safely ignored.

In the body of a passage, there are 3 different types of things to deal with: text, links, and commands. When a passage is being displayed, text should appear as typed (including spacing), however, links will display differently than they appear in the input file, and commands can cause a variety of different things to happen.

A.2 Link

Links in the body will be denoted by double brackets: `[[` and `]]`. Links are treated differently depending on whether or not they contain the characters `-&gt;`. A link that doesn't contain these characters should appear in the text without the double brackets, and this link should be presented to the reader as an option to further the story after the passage has displayed. When selected, the link should display the passage with the name that matches the link text.

A link that contains the characters `-&gt;` should display as the characters to the left of `-&gt;`; however, it should link to the passage whose name matches the characters to the right of the `-&gt;`.

Example links:

`[[Simple]]`

Displays as "Simple"; links to passage named "Simple"

`[[Take the blue pill-&gt;Bad dream?]]`

Displays as "Take the blue pill"; links to passage named "Bad dream?"

A.3 Commands

Commands are denoted by a single word and a colon immediately after an open parenthesis. Your IF interpreter should support 5 different commands, `(display:`, `(set:`, `(if:`, `(else-if:`, and `(else:`.

A.3.1 Display command

The display command should be replaced by the text of the passage with a given name. The name of the passage will appear between two copies of `&quot;` inside of the parentheses with `display`.

Example display command:
`(display: &quot;Status&quot;)`
Displays the passage named "Status".

Note: links in the displayed passage should be included as possible options to the reader, and commands should activate as normal.

A.3.2 Set command

The `(set:` command allows the IF author to define and set the value of a variable. Note that the `(set:` command will never display any text; however, it will execute any time a passage containing it is displayed. Variables that do not exist are created, while variables that do exist are updated. While the full specification for Harlowe allows for three different types of variables (numeric, string, and Boolean), you should treat all variables as Boolean.

The first word after the colon in the `(set:` command will be a variable name, which always starts with `$`. The second will be the key word `to`, and the third will be the assigned value (`true` or `false`). [1]

Example set command:
`(set: $ateCake to true)`
Stores `true` *as the value of the* `$ateCake` *variable*

A.3.3 If/Else if/Else

The `(if:`, `(else-if:`, and `(else:` commands act much like they do in C++.

The `(if:` and `(else-if:` commands will be followed by a variable, the key word `is`, and a value to test the variable against, followed by the closing parenthesis. `(else:)` has no condition.

The blocks that if, else-if, and else apply to are denoted by brackets `[]`. Note that links and other commands (including other `(if` commands) may be embedded in these blocks.

Example if command

```
(if: $ateCake is true)[You are quite full.]
(else-if: $ateCookie is true)[You sigh contentedly.]
(else:)[You still feel a mite peckish.]
```

[1]While Harlowe allows much more sophisticated versions of this syntax, including arithmetic operations (+, −, etc.), relational operators (<, <=, etc.), and logical connectives (and, or, not), we will only support simple assignments.

A.4 Running the interpreter

Your IF interpreter should start by opening the file `if.HTML` and reading in the story data, as described in the sections above. You may assume that the input does not contain any syntax or logical errors (e.g., testing a variable in an `(if:` command before it has been `(set:)`.

Once it has constructed objects to represent the structure of the story, the interpreter should start by displaying the first passage defined in *if.HTML*.

When displaying a passage, your program should execute any `(display:`, `(set:`, or `(if:` commands appropriately and properly display all links. Note that your interpreter will need to keep track of the variables that have been defined, as well as their values. You may use an `unordered_map` object for this purpose (`#include <unordered_map>`); you do not need to define a specialized class to match variables to their values. We will cover `unordered_map`s more when we discuss the Standard Template Library in class.

After displaying a passage, the interpreter should print out a numbered list of all links in the passage, and prompt the user to select one. This list should start numbering at 1; e.g.,

```
1. Take the red pill
2. Take the blue pill
```

Once the reader has selected a link, your interpreter should display the corresponding passage (see Links section above). This should continue until the reader reaches a passage with no links, at which point the interpreter should terminate.