ESP / Ass2 WS17

From Progwiki
< ESP

contents

text Adventure

In the context of the second practice example, a simple computer game (a multiple choice adventure - similar to a game book) will be created. This is a written story on the course of which the reader can influence by choosing from several decisions.

The story consists of individual elements. Each of these elements ends with a selection (choice A or choice B), with which the player decides how the story goes on. So there are in the end different possible ends that the player can reach.

featured

Command line parameters

The file name for the start file is transferred as a command line parameter.

To call the program, the following command is used:

```
./ass2 [filename]
```

[filename] is the file name of the start element. Wrong entries or files should be treated as described under Return values and error messages .

example

```
./ass2 start_of_story.txt
```

Note: The file extension * .txt does not have to exist for a file to be valid.

file format

The elements of the game are saved as individual text files. Each text file is structured as follows:

```
[Title of the item] \ n
[Filename dial A] \ n
[Filename dial B] \ n
[View Text]
```

The first line stores the title of the element (see Data Structure and Output). The second and third lines each contain a file name of a file in which the next element can be found for the respective selection option. If there are no further choices in the current element (because you have reached the end of the story), **both of** these lines will consist of:

```
- \ n
```

There is no case for which only one of the choices for another file is used (and the other is an endpoint).

From the fourth line follows the text of the current element (which itself may consist of any number of lines).

Any deviation from this structure means a corrupt file (see error messages).

example file

```
Chapter 1
chapter_21.txt
chapter_42.txt
'Would you tell me, please, which way I ought to go from here?'
'That depends on a good deal on where you want to get to,' said the Cat.
'I do not care much where -' said Alice.
'Then it does not matter which way you go,' said the Cat.
'- so long as I get SOMEWHERE,' Alice added as an an explanation.
'Oh, you're sure to do that,' said the Cat, 'if you only walk long enough.'
```

data structure

Design a structure to store the individual elements of the story. These elements must be able to cover the following contents:

- a C string for the title
- a C string for the text
- a reference to the element for choice A
- a reference to the element for choice B

circular references

In the basic version, which you have to implement for the levy, no circular references can occur between the individual structures (in the simplest case: element 1 refers to element 2 and this again to element 1). ~~So there is only one way to each element~~ , the individual elements form a binary tree.

Bonus task

For the support of county references (a choice can bring the player back to the game earlier), you will receive up to 5 bonus points (depending on the quality of your implementation). Bonus points should include the following to get the full 5 points:

- Recognize a referral to yourself.
- Recognize circular references in the tree.
- Realize that there is a circle but also an end and thus start the game normally.
- Detect when a file occurs several times and then merge them (symlinks, copies, other paths, etc.)

output

For each element, the output is as follows:

```
------------------------------ \ n
[Title of the item] \ n
\ n
[View Text] \ n
\ n
Your choice (A / B)?
```

Notes: Note the space to complete. The first line consists of 30 '-'.

In the event that an end of the story has been reached, the output is as follows:

```
------------------------------ \ n
[Title of the item] \ n
\ n
[View Text] \ n
\ n
End up
```

The program will be terminated with the return value 0 in this case.

input

After the element has been output, the user must enter "A \ n" or "B \ n". The corresponding next element is output (see above).

When entering an EOF, the program should be terminated without further output.

error messages

Order of error messages

The priority of the error messages corresponds to the order in which they are called. If several error messages apply to an input, only the most important one should be output and the command subsequently aborted.

Return values and program-terminating error messages

- If the program is called incorrectly (eg too many arguments):

```
Usage: ./ass2 [file-name] \ n
```

  - Return value: 1

- If the memory runs out during runtime, the following should be displayed and returned:

```
[ERR] Out of memory. \ N
```

  - Return value: 2

- If an error occurs when reading in the files at program start (corrupt or non-existent file), the program must be aborted as follows:

```
[ERR] Could not read file [filename]. \ N
```

  - Return value: 3

Non-interruptible error messages

- Should the user enter anything other than A or B when entering his choice.

```
[ERR] Please enter A or B. \ n
```

  - Then read again the choice.

specification

limitations

- All files must already be read in at program start (ie no successive read-in depending on the input of the player)
- no further issues
- all issues must be done on stdout

Allowed libraries

- all C standard libraries

delivery

- Filenames according to tax list
- Archive does not contain directories or other files
- Delivery by latest 7.12.2017 14:00:00 clock

made list

- Source code (ass2.c) in a .tar.gz or .zip archive (ass2.tar.gz or ass2.zip)

Responsible tutors

- Florian Bernhardt
- Marcel Nageler