FIT1008 – Intro to Computer Science
Assessed Prac 2 – Weeks 8 and 9

Semester 2, 2018

Objectives of this practical session

To be able to implement and use basic containers in Python.
Note:

- **You should provide documentation and testing for each piece of functionality in your code. Your documentation needs to include pre and post conditions, and information on any parameters used.**
- **Create a new file/module for each task or subtask.**
- **Name your files task[num]_[part] to keep them organised.**

Assessment: During your practical class in week 9
Submission: via Moodle, immediately after assessment

Testing

For this prac, you are required to write:

(1) **a *function to test* each function or method you implement, and**

(2) ***at least two test cases* per function. There is no need to test *menu* functions, but all ADT operations should be tested separately.**

The cases need to show that your functions or methods can handle both valid and invalid inputs.

Task 1 [9 marks]

Implement a complete version of an Array-Based List. Use 50 as the maximum number of elements. To create arrays use the method `build_array` from `referential_array.py` as discussed in the Lectures. Your List should include implementations for the following 10 functions:

- `__str__(self)`: Returns a string representation of the list. Structure the string so that there is one item per line. Called by `str(self)`
- `__len__(self)`: Returns the length of the list. Called by `len(self)`
- `__contains__(self, item)`: Returns True if item is in the list, False otherwise. Called by `item in self`
- `__getitem__(self, index)`: Returns the item at `index` in the list, if `index` is non-negative. If it is negative, it will return the last item if `index` is -1, the second-to last if `index` is -2, and so on up to minus the length of the list, which returns the first item. The function raises an IndexError if `index` is out of the range from `-len(self)` to `len(self)`. Called by `self[index]`
- `__setitem__(self, index, item)`: Sets the value at `index` in the list to be `item`. The `index` can be negative, behaving as described above. Raises an IndexError if `index` is out of the range from `-len(self)` to `len(self)`. Called by `self[index] = item`

- `__eq__(self, other)`: Returns True if this list is equivalent to `other`. Called by `self == other`
- `append(self, item)`: Adds `item` to the end of the list. Remember the underlying array is and should remain of fixed size. The operation should raise an Exception if the list is full.
- `insert(self, index, item)`: Inserts `item` into `self` before position `index`. The `index` can be negative, behaving as described above. Raises an IndexError if `index` is out of the range from `-len(self)` to `len(self)`.
- `remove(self, item)`: Deletes the first instance of `item` from the list. Raises a ValueError if `item` does not exist in `self`
- `delete(self, index)`: Deletes the item at `index` from the list, moving all items after it towards the start of the list. The `index` can be negative, behaving as described above. Raises an IndexError if `index` is out of the range from `-len(self)` to `len(self)`.
- `sort(self, reverse)`: Sorts the items in the list in ascending order if `reverse` is `False` or descending order if is reverse is `True`. Pick your favourite sorting algorithm from those covered in the Lectures.
- Make your class iterable.

Task 2 [2 marks]

Modify your list implementation so that the size of the underlying array is dynamic. The base size of the array is 20 and should never be less than 20. However, if the list becomes full, it is resized to be 2 times larger than the current size. Likewise, the underlying size should decrease by half if the underlying array is larger than the base size but the content occupies less than $\frac{1}{8}$ of the available space. When resizing the list, retain the contents of the list. That is, when it is initially filled, it will be resized to 20 items, then 40, while retaining the contents initially in it. The same happens when the size of the array shrinks.

Task 3 [2 marks]

Re-do task 1, now with a Linked Structure instead of an array.

Task 4 [2 marks]

- Implement a function that takes a filename as input and reads it into an instance of the class you implemented on Task 2. For each line in the file, store it as a single item in the list. [1]
- Implement a function that takes a filename as input and reads it into an instance of the class you implemented on Task 3. For each line in the file, store it as a single item in the list.

[1] For a refresher on how to read data from a file, read the tutorial found at `https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files`

CHECKPOINT

(You should reach this point during week 8)

Background

The editor `ed` was one of the first editors written for UNIX. In this prac we will use the Array-Based list to implement a version of a line-oriented text editor based on `ed`. The text editor `ed` is very similar to the common UNIX text editors `vi` and `vim` (it is in fact their predecessor). [2]

Important: Our commands will be different from the `ed` commands.

[2] find out more about `ed`, you can read the man page (by typing: `man ed` into a linux or MacOS X terminal), or visit, for example: `http://roguelife.org/~fujita/COOKIES/HISTORY/V6/ed.1.html` or `https://youtu.be/BNYpmLH6IjQ` (YouTube tutorial)

To implement a simple line-oriented text editor, the idea is as follows. Suppose a file contains the following lines:

```
Yossarian decided
not to utter
another word.
```

where the string "Yossarian decided" is considered to be in line 1, "not to utter" is considered to be in line 2, etc. We want to store every line in the file in a data type that allows users to easily manipulate (delete/add/print) any line by simply providing the line number they want to modify. This means we should use a list data type (as opposed to a stack or a queue).

Task 5 [6 marks]

Write a text editor as a Python program that allows a user to perform the 6 commands shown below using a menu. It is advisable that the Editor itself is a class with an attribute containing an list of lines. The list should be the type you have implemented on Task 2.

insert num: which inserts a line of text (given by the user) in the list before position *num*, and raises an exception if no *num* is given

read filename: which opens the file, *filename*, reads all the lines in from the file, put each line as a separate item into a list, and then closes the file.

write filename: which creates or opens a file, *filename*, writes every item in the list into the file, and then closes the file.

print num1 num2: which prints the lines between positions *num1* and *num2*, if $num1 < num2$.

delete num: which deletes the line of text in the list at position *num*, and deletes all the lines if no *num* is given.

search word: which takes a *word* and prints the line numbers in which the target word appears. Search should be case insensitive. Search should work as expected in a standard text editor, for example ignoring punctuation. [3] This function must be accessible through the menu via commands `search`.

[3] Documentation for Python string functions can be found at `https://docs.python.org/3/library/stdtypes.html`

quit: which quits the program.

Important: All errors should be caught and a question mark, **?**, should be printed when an error occurs. Negative number lines are possible and should be handled following the convention described for the original list implementation. For example, *num = -1* refers to the last line of the text.
Tip: When you are testing your code, it is handy to have a source of reasonably large text files that you can use as test data. There is a huge repository of public domain text files at Project Gutenberg's websites. The Australian Project Gutenberg repository is at `http://gutenberg.net.au`. You are encouraged to download a couple of ebooks from here and use them to make sure your code can deal with large files. Be sure to download plain-text format, though – your program is not expected to deal with other formats.

Task 6 [4 marks]

Re-implement your editor using a Linked List instead of an Array-based list. If done properly, switching your text-editor to a Linked List implementation should take no more than changing a single line. To ensure this, make sure that the List ADT is implemented with the same functions and function signatures in both implementations (Tasks 1, 2, 3). Write a page of text, using what you know of theory to analyse how the performance of your text editor would change when switching implementations.