CSE 302 Assignment 2

Due: 2/24/23

---

1. This problem deals with List ADTs and implementations. *(30 pts)*

Choose an appropriate List ADT and implementation (between `AUList`, `LLUList`, `ASList`, and `LLSList`) for each of the list-based applications in parts a-c below. Explain your reasoning in 2-4 sentences for each.

a. A user wants to keep track of a list of product names such that

i. Most added names are subsequently removed before many additional names are added.
ii. Names deep in the list have been "adopted" for regular use and are almost never deleted or searched for in practice, while names early on are searched for and deleted frequently.
iii. The number of products in the list is unknown and unpredictable (could be 100 or 1 million).

b. A user wants to keep track of a list of celebrity pets by name such that

i. A stable quantity of names are infrequently changed (adding or removing) from the database.
ii. Searches for names occur thousands if not millions of times each.

c. A user wants to keep track of a list of his/her favorite song titles at any one time such that

i. Titles are added and removed very often, while individual title searches are extremely rare.
ii. The order of titles in the list is fairly unimportant.
iii. The maximum number of titles in the list is known in advance. When this number is reached the user generally chooses to completely discard the list entirely and start from scratch.

> **Note**
>
> While the `Stack` and `Queue` ADTs may be preferable in some of the above scenarios, do *not* include them in your answers. Also assume any array – if used – must have fixed size. *Make sure you justify your choices*!

---

2. This problem deals with various concepts we have discussed in class. *(30 pts)*

For each pair in *a-c*, define the two terms, and explain their relationship (similarities and differences, as appropriate). Your responses should be 3-6 sentences per part.

a. Sorted List ADT and Binary Search
b. The Stack ADT and Linked-list based Stack
c. The Queue ADT and Fixed-Front Array Queue

> **Note**
>
> Your response to each should include at least one detail regarding *operation efficiency* relevant to the topics, using *Big O Notation*.

3. This programming problem deals with the template-based stack `StackType` and queue `QueType` implementations, and strings. *(40 pts)*

You *must* use the provided `StackType.h` and `QueType.h` found in the provided *TemplateStackDriver.zip* and *TemplateQueueDriver.zip* files. Also, be certain to adhere to the example below in using the standard library `std::string` as input arguments not a character array (`char[] or char*`) as input, since we will test it using strings.

You are to create a function `decode` in a file **stringdecoding.cpp** as:

```
std::string decode(std::string exp, std::string code){}
```

The function should iterate through each character in `exp` (see std::string at function) and take one of two actions:

i. Characters in `exp` that are **not** also in the string `code` should be present in the output in normal, sequential order as they are encountered, without modification.
ii. All characters in `exp` that **are** also present in the string `code` should be present in the output *after the characters printed in part i.* AND **entirely in reverse order**.

> **Note**
>
> Recommended is to use the std::string find function to determine whether or not a character is in the *code* string. **(The autograder may not accept the *contains* function.)**

Sample input and output for reference is as follows:

```
decode("czitqommta_ehmumt_nio_szozir_eulopupoa_yeht_","_acefhilnpst")

// should return the following string:
"zqommmumozozruouoy_the_apple_is_in_the_attic"
```

As alluded to above, the recommended approach is to:

1. Create `StackType<>` and `QueType<>` instances of the appropriate types and add characters to them until the end of string `exp` is reached.
2. Put together the output string using appropriate operations.

> **Note**
>
> You are free to use helper functions as desired; however, all your code must be included in the **stringdecoding.cpp** file. You must not modify the `StackType` or `QueType` files, rather you need to include the `StackType` and `QueueType` into the **stringcoding.cpp** file.

> **⚠ Initializing Variables**
>
> Remember to initialize your variables! You can not assume that `int i;` will set `i` equal to 0. Instead, always set your variables to a value manually. Like so: `int i = 0;`

Instructions for Submission:

As before, submit your files to Gradescope.

- Responses to problems *#1* and *#2* should be in **.txt** or **.pdf** format only!
- The code for *#3* should be submitted as one single file, **stringdecoding.cpp**.
- You will be allowed 10 submissions.