Assignments

Assignment 2 - Returned

Title	Assignment 2
Student	Ryan Coslove
Submitted Date	
Grade	**5.00 (max 12.00)**
History	

Instructions

Download arraylist.c and arraylist.h.

1. (3 points) Extend arraylist by adding a function al_insert:

```
int al_insert(arraylist_t *list, int index, int item);
```

`al_insert(l,i,n)` adds a new item n into the list l at index i. Any items at or after index i in the list will be moved forward one space.

If i exceeds the current number of entries in the list, then i will become the new final element of the list. Any newly created entries between the previous last entry and i are not initialized and will contain undetermined values.

If data is not long enough to contain the new list, then either double its length or extend its length to include index i, whichever is longer.

That is,

```
al_init(&A, 10);       // create with length 10
al_insert(&A, 10, 0);  // increases length to 20
al_insert(&A, 90, 0);  // increases length to 91
// only the indicies 10 and 90 are initialized
```

`al_insert` returns 0 if it successfully added the item to the list and 1 if it encounters an error, such as being unable to allocate memory.

Attach your modified copies of arraylist.h and arraylist.c.

2. (6 points) Using arraylist as an model, create a library for string buffers (an array list containing characters). Rename the types and functions as appropriate (e.g., "arraylist" becomes "strbuf"). Include functions

corresponding to all the functions in arraylist, including `al_insert`. Ensure that the string held in `data` is null-terminated.

Add the following function:

```
int sb_concat(strbuf_t *sb, char *str);
```

`sb_concat` adds the string str to the end of the string held in sb. Assume that str is a null-terminated C string. Return 0 if successful, and 1 otherwise.

Note that `sb_append`, `sb_insert`, and `sb_concat` will share similar code to extend the length of the data. Consider putting this code in a separate helper function.

Attach strbuf.h and strbuf.c.

3. (3 points) Consider the behavior of the following code fragments

```
// 1
char *buffer;
...
for (i = 0; i < N; ++i) {
    strcat(buffer, input[i]);
}



// 2
strbuf_t *buffer;
...
for (i = 0; i < N; ++i) {
    sb_concat(buffer, input[i]);
}
```

Assume that sufficient memory has been allocated in both cases, that input contains at least N elements, and that all strings are null-terminated. Will there be any notable differences in the performance of these code fragments?

Enter your answer in the text box, or attach a text file.

Submitted Attachments

- arraylist.c (2 KB; Feb 25, 2021 9:42 pm)
- arraylist.h (1 KB; Feb 25, 2021 9:42 pm)
- strbuf.c (2 KB; Feb 25, 2021 9:43 pm)
- strbuf.h (1 KB; Feb 25, 2021 9:43 pm)
- Answer3.txt (1 KB; Feb 25, 2021 9:43 pm)

Additional instructor's comments about your submission

Part1: 2   Doesn't move all items forward correctly when insert
Part2: 2 not finished
Part3: 1 O(n^2) vs O(n)

Back to list

- For Student, Faculty and Staff: Contact the Office of Information Technology: help@oit.rutgers.edu: 833.OIT.HELP

  Report an Accessibility Barrier

- Rutgers University
- Copyright 2003-2022 The Apereo Foundation. All rights reserved. Portions of Sakai are copyrighted by other parties as described in the Acknowledgments screen.