COMP2521 Assignment 1 Specification

Textbuffer Interface

Jump to FAQ | z.afzal@unsw.edu.au

Your task is to implement an abstract textbuffer data type that meets the given interface. In addition, you will have to code to test your implementation. Ensure that you also test for boundary conditions and for memory management bugs.

You will submit the C code implementing the textbuffer ADT and your testing code.

Below this page describes the interface of textbuffer ADT that you are to implement. For your implementation, download `textbuffer.h` below and implement the type `TB` as well as all the functions whose prototype is given in the header file. All your code should go into a file `textbuffer.c`, which you have to submit to complete the assignment.

Changelog

Thursday 23rd August: Released to the poor poor students
Friday 24th August: Updated compilation flag from -std=c99 to std=gnu99
Saturday 25th August: Fixed typo in searchTB example, and in dumpTB comments in textbuffer.c and textbuffer.h. Updated addSuffix to addPrefix + added more examples
Monday 27th August: Added information on memory leak style marks. Added examples to formRichText, Updated FAQ, Clarified Empty/NULL Cases, Clarified SearchTB
Tuesday 28th August: Fixed mixedup terms in mergeTB empty case FAQ
Tuesday 28th August: Fixed empty case FAQ yet again. Clarified formRichTest spec. Added FAQ item for empty line input for newTB
Sunday 2nd September: Added case sensitive to searchTB behavior, fixed number to showLineNumbers in dumpTB, added formRichTextTB FAQ item, removed confusing term

from formRichTestTB Outline.

Tuesday 4th September: Added formRichTextTB FAQ item.

Saturday 8th September: Updated blackbox text commenting FAQ item

Sunday 9th September: Made gcc line copy and pasteable

Marks

The assignment is worth 10 marks. The marks breakdown is as follows:

Component	Mark
Autotesting of functionality	6
Testing	2
Subjective evaluation of style	2

Automarking - 6 Marks

We will run your textbuffer.c implementation on a number of tests. These will be much more comprehensive than the tests we run during submission. You get marks for each test you pass.

Testing - 2 Mark

You will create a suite of blackbox tests and whitebox tests. We will not actually be running your whitebox tests, your tutor will be marking them subjectively, but please make sure they compile. You should write your whitebox tests in your textbuffer.c file. These whitebox tests will be worth 1 mark.

You will also create a testTextBuffer.c file, that will contain your black box tests. These tests will be run against some of our own correct and also incorrect implemetations of textbuffers. Your test file should be able to pick up our errors, but pass our correct implementations. Your tutor will also subjectively assess your tests. All together your blackbox tests will be worth 1 mark.

Style - 2 Marks

Style marks will include comments, indentation, variable names etc and will also include marks for choosing an appropriate representation for your ADT a nd for efficiency of the algorithms you choose. For example, you will lose marks if your implementation of a function has work complexity of `O(n^2)` when there is a solution with `O(n)` or `O(n * log n)`

In addition style marks will reflect if your program has any memory leaks (memory you have allocated and have responsibility to free but never free'ed). Your program will be tested for memory leaks via `valgrind`

Plagiarism

This is an individual assignment. Each student will have to develop their own solution without help from other people. In particular, it is not permitted to exchange code or pseudocode. You are not allowed to use code developed by persons other than yourself. If you have questions about the assignment, ask your tutor.

Plagiarism is defined as using the words or ideas of others and presenting them as your own. UNSW and CSE treat plagiarism as academic misconduct, which means that it carries penalties as severe as being excluded from further study at UNSW. There are several on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Procedure

Make sure that you read and understand these. Ignorance is not accepted as an excuse for plagiarism. In particular, you are also responsible that your assignment files are not accessible by anyone but you by setting the correct permissions in your CSE directory and code repository, if using. Note also that plagiarism includes paying or asking another person to do a piece of work for you and then submitting it as your own work.

UNSW has an ongoing commitment to fostering a culture of learning informed by academic integrity. All UNSW staff and students have a responsibility to adhere to this principle of academic integrity. Plagiarism undermines academic integrity and is not tolerated at UNSW. Plagiarism at UNSW is defined as using the words or ideas of others and passing them off as your own.

If you haven't done so yet, please take the time to read the full text of

- UNSW's policy regarding academic honesty and plagiarism

The pages below describe the policies and procedures in more detail:

- Student Code Policy
- Student Misconduct Procedure
- Plagiarism Policy Statement
- Plagiarism Procedure

You should also read the following page which describes your rights and responsibilities in the CSE context:

- Essential Advice for CSE Students

Submission

Deadline: 11:59pm, Friday 14 September 2018.
Submission Details: TBA

Files

1. `textbuffer.h`

1. `textbuffer.c`

1. `testTextbuffer.c`

Note When we test your assignment it with be compiled with gcc and the following flags

```
gcc -Wall -Werror -std=gnu99 -O -lm -o textbuffer testTextBuffer.c textbuffer.c
```

ADT Specification

The following is a description of the components of the interface.

As marks are awarded by an automated marking program, you must follow this specification precisely. Otherwise, you risk getting few or no marks! You must **NOT** modify the `textbuffer.h` file.

The ADT type

We represent the ADT by way of a handle of type `TB`. The handle type is declared in the header file, but you will have to provide an implementation of the handle representation - i.e. of `struct textbuffer` - as part of your implementation:

```
typedef struct textbuffer *TB;
```

Refer to the lecture about ADTs for examples of this construction.

Required properties of the implementation

A textbuffer is an ordered collection of strings, where each string represents one line of a text file. Your implementation must keep the lines of a textbuffer in a linked data structure (such as a linked list or a variant of that). Each line must be represented as a (dynamically allocated) character array. Adding, deleting, or moving of lines requires manipulation of the linked structure. Such a data structure may, for example, be used as part of a text editor.

Constructor and destructor

Make a new TB

The function `newTB` allocates a new textbuffer and initialises its contents with the text given in the array. The lines in the input array are all terminated by a `'\n'`. The whole text is terminated by a `'\0'`.

```
TB newTB (char text[]);
```

Destory a TB

The function `releaseTB` frees the memory occupied by the given textbuffer. It is an error to access a buffer after freeing it.

```
void releaseTB (TB tb);
```

Query functions

The following functions do not alter their textbuffer argument.

Allocate and return an array containing the text in the given textbuffer

Each individual line of the textbuffer needs to be terminated by a `'\n'` (this includes the last line). The whole text must be `'\0'` terminated. It is the caller's responsibility to free the memory occupied by the returned array. If there are no lines in the textbuffer, return `the empty string`. If and only if `showLineNumbers` is true additonally append to each line before writing to the return array the line number proceeding by a dot and a space. I.e `hello world` would be returned as `1. hello world`

```
char *dumpTB (TB tb, int showLineNumbers);
```

Return the number of lines of the given textbuffer.

```
int linesTB (TB tb);
```

Textbuffer editing

For all editing functions, if any of the arguments indicates a line number is out of range (i.e., smaller than zero or bigger than the number of lines in the buffer minus one), the function has to print a suitable error message and terminate the program with the standard function `abort()`.

Add the supplied prefix to all lines between `pos1` and `pos2`

The first line of a textbuffer is at position 0.

`void addPrefixTB (TB tb, int pos1, int pos2, char* prefix);`

If `pos2` is less than `pos1`, abort

Consider calling `addPrefixTB(tb,0,2,"goodnight ")`

```
+ ---------------------------- +         + ------------------------------------ +
| room                         |         | goodnight room                       |
| moon                         |  --->   | goodnight moon                       |
| cow jumping over the moon    |         | goodnight cow jumping over the moon  |
| light                        |         | light                                |
+ ---------------------------- +         + ------------------------------------ +
```

Remove the lines between and including from and to from the textbuffer `tb`. Free the memory of the deleted lines.

`void deleteTB (TB tb, int from, int to);`

If `to` is less than `from`, abort

Combining textbuffers

For all combining functions, if any of the arguments indicates a line number is out of range, the function has to print a suitable error message and terminate the program with the standard function `abort()`.

Note that in this case, if the number of lines in tb1 is n, then n is a valid argument for pos (the text is added after the end of the buffer).

Merge tb2 into tb1 at line pos

Afterwards what was line 0 of tb2 will be line pos of tb1. The old line pos of tb1 will follow after the last line of tb2. After this operation tb2 can not be used anymore (as if we had used `releaseTB()` on it).

```
void mergeTB (TB tb1, int pos, TB tb2);
```

Copy tb2 into tb1 at line pos

Like `mergeTB()`, but tb2 remains unmodified and is still usable independent of tb1.

```
void pasteTB (TB tb1, int pos, TB tb2);
```

Extracting textbuffers

For all extracting functions, if any of the arguments indicating a line number is out of range, the function has to print a suitable error message and terminate the program with the standard function `abort()`.

The textbuffers returned by the extracting functions are as newly created with `newTB()`.

Cut the lines between and including `from` and `to` out of the textbuffer `tb`.

The cut lines will be deleted from tb. If `to` is less than `from`, return `NULL`.

```
TB cutTB (TB tb, int from, int to);
```

Return a linked list of all matches in `tb` of a certain string

The search is **case sensitive** and the textbuffer `tb` must remain unmodified.

```
Match searchTB (TB tb, char* search);
```

consider calling `searchTB(tb,"love")` on the following TB

```
1 Hello World My
2 name is jarred lovegood
3 and i love carley ray jepson
```

this would give us a list:

```
+=================+    +=================+
|  lineNumber: 2, |    |  lineNumber: 3, |
|  charIndex: 15, |    |  charIndex: 6,  |
|  next: ------------->|  next: -------------> NULL
+=================+    +=================+
```

Note that the line number is **1 indexed** whereas the character index is **0 indexed** and refers to the first character of the match string
Note that Match is a pointer to the first node in the list

Rich text

The function `formRichText` searches every line of `tb` and performs the following substitutions

String	Replacement	Example
some string	<b>some string</b>	*hello* -> <b>hello</b>
some string	<i>some string</i>	_hello_ -> <i>hello</i>
#some string ...	<h1>some string ...</h1>	#hello -> <h1>hello</h1>

The matching is simplistic in that you would begin scanning at the first special character and continue to consume characters (ignoring any further special characters) until the matching special character. If there is no matching character, nothing is done and the next special character is processed

Note that the # character must be the first character in a line or else it does nothing. In addition it matches until the end of the line and not until a matching #. See example below.

Example	Result
*some string	*some string
*some string*lol*	<b>some string</b>lol*
*some_string*again_	<b>some_string</b>again_
some _string_	<b>some</b> <i>string</i>
some *string_again_	some *string<i>again</i>
some#string*once_again*	some#string<b>once_again</b>
#string_stuff_	<h1>string_stuff_</h1>

In the case of nested special characters, i.e

```
*some_string_*
#some _string_
```

You may take the outermost element and ignore any nesting.

Example	Result
*some_string_*	<b>some_string_</b>

Example	Result
`#some _string_`	`<h1>some _string_</h1>`

```
void formRichText (TB tb);
```

Assignment 1 Bonus Challenges

Differences between two textbuffers (1 mark)

Given two text files, we sometimes want to know what changes are made from one file to another file.

The function `diffTB` works out which lines of texts are added or deleted from `tb1` to get `tb2`. The returned string of the function is an edit solution consisting of a series of add and delete commands. Applying such commands on `tb1` in sequence should result in `tb2`.

```
char* diffTB (TB tb1, TB tb2);
```

An edit solution should have one command per line to either add or delete a line of text at a specific line number. An example is given below. The first command adds a line of text 'add this line please' at line 2 of the current textbuffer (counting from 0). The existing line 2 is moved to line 3, and so on. The second command deletes the line 3 of the textbuffer. The last command adds the specified text at line 12 of the textbuffer.

```
+,2,add this line please
-,3
+,12,add this line as well please
```

A mark is given if your solution satisfies two criteria given below:

- *Correctness* - applying your edit solution on tb1 results in tb2.
- *Compactness* - the size of your edit solution (i.e. number of lines) is smaller than or equal to the size of our model solution. This is to avoid trivial solutions like delete all

lines of tb1 and add all lines of tb2.

Undo and redo operations (1 mark)

The function `undoTB` should be able to reverse at most 10 recently called operations on `tb`. Applicable operations are, `deleteTB`, `mergeTB`, `pasteTB` and `cutTB`. Each time `undoTB` is called, one operation is reversed on `tb`. When the maximum number of allowable undo operations is reached, nothing is done on `tb`.

```
void undoTB (TB tb);
```

The function `redoTB` calls operations that are reversed by `undoTB` again in order. Similar to `undoTB`, this function should redo one operation on tb per function call. However, when a new operation is called on `tb`, any reversed operations cannot be executed again with `redoTB`.

```
void redoTB (TB tb);
```

COMP2521 assn1 textbuffer FAQ

General questions

Can I modify `textBuffer.h`?

No.

Can I add my own structs and functions to textbuffer.c?

Yes! Make sure functions are declared `static`, and you document what the functions and structures you add are for.

Can I use functions from `<string.h>`?

Yes. It's much, much harder if you don't.

Will I need to check my code with Valgrind?

We'll certainly be checking your submission with Valgrind for memory leaks.

Can `TB` be defined like `link` in the lecture examples?

If `TB` points directly to the head of the list, functions like `mergeTB` cannot function correctly, as they may change the head of the list.

Can TB ever be NULL?

> it can be in the case something goes wrong but in any case you have a `NULL` TB the correct logic is to abort and print a suitable error message

If i abort should i free any memory?

> This won't be tested but it's good practice that if you created any memory before hitting the abort case you free it before crashing.

newTB

How does `newTB` work?

> If the input text is, for example, `"Hi\nhow\nare\nthings\n\0"`, the buffer should contain the following lines: `{ "Hi", "how", "are", "things" }`. You will have to process the input string, extract all the substrings separated by newline, and copy them into the entries of your buffer structure.

Should I leave the `'\n'` characters in?

> Depending on your approach to splitting text, they may already be gone. The only other place you need the `'\n'` characters is in `dumpTB`, so you could probably get away without storing them. But it is up to you.

Is it safe to assume that the text will always have a new line at the end?

> Yes, text will always have a newline.

What should happen with multiple consecutive newlines?

> Each newline marks a new node in the text buffer. You need to track empty lines.

Can i assume a max size for lines?

No. Your program should be able to dynamically create memory needed for your character arrays depdening on input.

What if the input text is a empty string?

create a empty TB.

What if the input text is just the newline character?

create a TB with 1 empty line

Can I use *strtok(3)*

Yes but it does not work with consecutive delimiters so it will not be very helpful. You can however use `strsep` instead which can!

Note, that to use strtok or strsep, the input string needs to be mutable and this isn't guaranteed in the spec.
Hint: But you can make a mutable copy and then use strtok or strsep

ReleaseTB

How should I write tests for `releaseTB`?

You cannot. You can't write a black-box test for a destructor.

When you `free(3)` memory, all you're saying is that you no longer need the block of memory you had a pointer to; it should be irrelevant to you whether that memory's value changes or becomes invalid in some way, because **YOU ARE ABSOLUTELY FORBIDDEN FROM ACCESSING THE MEMORY ONCE FREE'D**. Use after free is an illegal and undefined operation. You have no way to invalidate the pointers (read: change any values outside your ADT, including outside pointer references to its state structure).

A good test that your `releaseTB` worked is that your program is still running after you do so.

And then you can use valgrind to check for memory leaks.

DumpTB

My textbuffer has no lines; what should `dumpTB` return?

`""` i.e empty string

AddPrefixTB

Can the input string have new lines in it?

No. We will not test these cases

Can the input string be the empty string?

Yes, in this case do nothing

Can the input string be the NULL?

No, in this case abort

searchTB

Can the input string have new lines in it?

No. We will not test these cases

Can the input string be the empty string?

Yes, in this case return a empty list

Can the input string be the NULL?

No, in this case abort

How do you handle the case where the input is a repeated pattern e.g. looking for 'abab' in 'ababab'

Match greedily. i.e ababab returns (1,0) only. ie. it matches just the first instance. **abab**ab

formRichTextTB

How should i handle empty string cases such as `**`

In this case nothing should happen, only add the tags if there is at least 1 character being acted on

Should i match over multiple lines?

No.

MergeTB

What should happen if I `mergeTB (tb1, 1, tb1)`?

Attempts to merge a textbuffer with itself should be ignored.

Should I call `releaseTB` as well?

> **No!** This will probably destroy both the source and destination textbuffers. However, you've moved the contents of the source textbuffer, so you can just *free(3)* as you would in `releaseTB`. You must not subsequently dereference it; that's a use-after-free and (say it with me, folks!) use after free is illegal.

Can I concatenate text buffers with `mergeTB`?

> The correct behaviour should be as follows, for `mergeTB (dest, pos, src)`:
>
> - `pos=0`: insert `src` before the start of `dest`.
> - `pos=linesTB(dest)-1`: insert `src` before the last line of `dest`.
> - `pos=linesTB(dest)`: append `src` at the end of `dest`.

What should happen if `tb1` or `tb2` are empty?

> Both may be empty. If the destination is empty then `pos == 0 == linesTB(dest)` causing the source to be appended to the end of the empty TB.

diffTB

Does `diffTB` change either of its textbuffer arguments?

> No. `diffTB` is non-destructive.

undoTB and redoTB

What should happen if i undo a merge? is `tb2` alive again?

> If you ran `mergeTB(dest,pos,source)` source is now dead and calling `undoTB(source)` is invalid. Calling `undoTB(dest)` Should have expected behaviour

testTextBuffer

> **Will I be marked on writing tests for the bonus functions?**
>
> > No. You should write tests for the bonus functions if you complete them for your own benefit, but you should comment them out in the testTextBuffer blackbox tests

for submission

Is it ok if my textbuffer does not pass my testTextBuffer tests?

Yes! This will happen if you are not able to complete all functions in textbuffer.c properly. This is ok. But it should compile!

Should I write a main in textbuffer.c to call my whiteboxtest function or call it from the testTextBuffer file?

You can do either, as long as you comment out the main, or the call to the whiteboxtext function for submission. Note that you don't have to comment out the actual white box test function, just any call to them.