

CMPT125, Fall 2021

Homework Assignment 5

Due date: Friday, December 2, 2022, 23:59

You need to implement the functions in **assignment5.c**.
Submit only the **assignment5.c** file to Canvas.

Solve all 3 problems in this assignment.

Grading: The assignment will be graded automatically.

Make sure that your code compiles without warnings/errors, and returns the required output.

Compilation: Your code MUST compile in CSIL with the Makefile provided.

If the code does not compile in CSIL, the grade on the assignment is 0 (zero).

Even if you can't solve a problem, make sure the file compiles properly.

Warnings: Warnings during compilation will reduce points.

More importantly, they indicate that something is probably wrong with the code.

Dynamically allocated arrays: Do not use variable length arrays! Never!

If you need an array of unknown length, you need to use malloc.

Memory leaks: Memory leaks during execution of your code will reduce points.

Make sure all memory used for intermediate calculations are freed properly.

Readability: Your code must be readable, and have reasonable documentation, but not too much. No need to explain `i+=2` with `// increase i by 2`.

Write helper functions if that makes the code more readable.

Testing: An example of a test file is included.

Your code will be tested using the provided tests as well as additional tests.

Do not hard-code any results produced by the functions as we will have additional tests.

You are strongly encouraged to write more tests to check your solution is correct, but you don't have to submit them.

1. You need to implement all the functions in **assignment5.c**.
2. You should not add `main()` to `assignment4.c`, because it will interfere with `main()` in the test file.
3. Submit only the **assignment5.c** file to CourSys.

Problem 1 [30 points].

Implement the following two functions that allow breaking a string into non-empty tokens using a given delimiter. For example,

- For a string “abc-EFG-hi”, and the delimiter ‘-’: the list of tokens is [“abc”, “EFG”, “hi”]
- For a string “abc-EFG---hi-”, and the delimiter ‘-’: the list of tokens is [“abc”, “EFG”, “hi”]
- For a string “abc”, and the delimiter ‘ ’: the list of tokens is [“abc”]
- For a string “++abc++”, and the delimiter ‘+’: the list of tokens is [“abc”]

That is, we break the string using the given delimiter, and the tokens are only the non-empty substrings.

1. [10 points] The function `count_tokens` gets a string `str`, and a char `delim`, and returns the number of tokens in the string separated by `delim`.

```
int count_tokens(const char* str, char delim);
```

For example

- `count_tokens("abc-EFG--", '-')` needs to return 2.
- `count_tokens("++a+b+c", '+')` needs to return 3.
- `count_tokens("****", '*')` needs to return 0.

2. [20 points] The function `get_tokens` gets a string `str`, and a char `delim`, and returns the array with the tokens in the correct order. The length of the array should be the number of tokens, computed in `count_tokens`.

```
char** get_tokens(const char* str, char delim);
```

For example:

- `get_tokens("abc-EFG--", '-')` needs to return [“abc”, “EFG”]
- `get_tokens("++a+b+c", '+')` needs to return [“a”, “b”, “c”].
- `get_tokens("****", '*')` needs to return either NULL or an empty array.

Remark: Note that the returned array and the strings in it must all be dynamically allocated.

Problem 2 [10 points]

In this question we get a string `str`, and an array of `n` chars: `c[0...n-1]`. The function returns an array of `n` strings consisting of the `str` concatenated with `c[0]`, `c[1]`... `c[n-1]`.

The returned array and the strings in it must all be dynamically allocated.

```
char** append_chars(const char* str, int n, char* chars);
```

For example

- `append_chars("abc", 3, ['x','y','c'])` needs to return the array [“abcx”, “abcy”, “abcc”].
- `append_chars("", 4, ['x','y','z','x'])` needs to return the array [“x”, “y”, “z”, “x”].

Problem 3 [30 points]

In this question we get a string consisting of digits 2...9, and need to return all words/strings that can be obtained on a phone using these digits. The translation is:

2 - a,b,c
3 - d,e,f
4 - g,h,i
5 - j,k,l
6 - m,n,o
7 - p,q,r,s
8 - t,u,v
9 - w,x,y,z

1. [10 points] The function `count_words` gets a string `phone_number`, and returns the number of distinct words that can be constructed from this `phone_number`.

```
int count_words(const char* phone_number);
```

For example

- `count_words("238")` needs to return 27.
- `count_words("2345677")` needs to return 3888.
- `count_words("7777777777")` needs to return 1048576.

2. [20 points] The function `get_words` gets a string `phone_number`, and returns the array of distinct words that can be constructed from this `phone_number`.

```
char** get_tokens(const char* str, char delim);
```

For example:

- `get_words("22")` will have the following words, and no other words: `["aa", "ab", "ac", "ba", "bb", "bc", "ca", "cb", "cc"]`.
- The order of the words can differ, but you need to return all these words, each word exactly once.
- `get_words("43556")` will have the words: "hello", "idklm", and many more words. There is no room here to write all of them.
- `get_words("7")` will return `["p", "q", "r", "s"]` in some order.

Remarks:

- Note that the returned array and the strings in it must all be dynamically allocated.
- You may assume that `phone_number` has length between 1 and 12.

Problem 4 [30 points]

For this problem use the following struct representing a node in a Binary Tree.

```
struct BTreeNode {
    int value;
    struct BTreeNode* left;
    struct BTreeNode* right;
    struct BTreeNode* parent;
};
typedef struct BTreeNode BTreeNode_t;
```

You may also use the functions of Binary Search Tree provided in the zip file.

Implement the following two functions on Binary Search Trees.

In both questions you may assume the tree is not null and not empty

1. [15 points] The function gets a binary search tree and returns the maximal number.

```
int get_max(BST_t* tree);
```

2. [15 points] The function gets a binary search tree and returns the median element in the tree. For example,

- If the tree consists of the numbers {1,3,6,8,100}, the output should be 6
- If the tree consists of the numbers {1,2,3,7,8,100}, the output should be 7.
- If the tree consists of the numbers {5,100}, the output should be 100.
- If the tree consists of the numbers {5,8,10}, the output should be 8.

If the number of nodes is n , and the numbers in the tree are $A[0 \dots n-1]$:

- If n is even, the function returns $A[n/2]$.
- If n is odd, the function returns $A[(n-1)/2]$.

```
int get_median(BST_t* tree);
```