

CMPT125, Spring 2023

Homework Assignment 4

Due date: Sunday, April 9, 2023, 23:59

You need to solve the first two problems in **assignment4.c**.
Submit **assignment4.c** to CourSys.

Solve all problems in the assignment.

The assignment will be graded automatically.

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

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.

The assignment contains the object file stack_int.o. This object file contains the functions used for Problem 4. This .o file is specific for the CSIL machines.

If you want to run your code on your machine, you will need to create an object file stack_int.o for you operating system.

This is done using the command “gcc -c stack_int.c” in the lib folder

Warning during compilation will reduce points.

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

Memory leak during execution of your code will reduce points.

Check that all memory used for intermediate calculations are freed properly.

Your code must be readable, and have reasonable documentation, but not too much.

No need to explain `i+=2` with `// increase i by 2`.

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 the functions in **assignment4.c**.
2. If necessary, you may add helper functions to the assignment4.c file.
3. You should not add `main()` to assignment4.c, because it will interfere with `main()` in the test file.
4. Submit only the **assignment4.c** file to CourSys.

Good luck!

Problem 1 [15 points]

Write a function that gets an array of strings of length n , and sorts the strings in the array. In the sorted array the strings are arranged as follows:

- Look only at the digits in the string. Compute the sum of all digits in the string.
- The string with the lower sum should appear first in the sorted array
- If two strings have the same sum, the two strings can be in any order

One way to solve it is to implement the comparison function, and then apply `qsort()` on the array with this comparison function.

```
void sort_strings(const char* A[], int n)
```

For example:

- Suppose `A = ["w00x00", "ab1", "6_@7123h", "80", "b012", "b"]`.
- After running `sort_strings()` `A` becomes either
`["w00x00", "b", "ab1", "b012", "80", "6_@7123h"]` or
`["b", "w00x00", "ab1", "b012", "80", "6_@7123h"]`.

Problem 2 [20 points]

Write the following variant of selection sort.

```
// The function gets an array of length n of ints,  
// and applies the selection sort algorithm on the array.  
// The function returns the array of ints of length n, where  
// output[i] is the index where the i'th smallest number was found  
int* selection_sort(int* A, int n);
```

For example: If the input is the array `A=[40,30,60,10,20,50]`, Insertion Sort on `A` works as follows:

- The minimum is 10 in the position `i=3`. The array becomes `[10,30,60,40,20,50]`.
- The next minimum is 20 in the position `i=4`. The array becomes `[10,20,60,40,30,50]`.
- The third minimum is 30 in the position `i=4`. The array becomes `[10,20,30,40,60,50]`.
- The next minimum is 40 in the position `i=3`. The array becomes `[10,20,30,40,60,50]`.
- The next minimum is 50 in the position `i=5`. The array becomes `[10,20,30,40,50,60]`.
- The last minimum is 60 in the position `i=5`. The array is now sorted: `[10,20,30,40,50,60]`.

After the execution of the function

(1) `A` will become `[10,20,30,40,50,60]`

(2) the function returns `[3,4,4,3,5,5]`.

Problem 3 [25 points]

Write a generic implementation of insertion sort.

```
// the function gets an array of length n of objects of given size  
// and a compare function  
// The function applies insertion sort on the array  
// using the compare function  
// if compare(a,b)<0, then a must come before b in the sorted array  
// if compare(a,b)>0, then a must come after b in the sorted array  
void gen_insertion_sort(void* array, int n, size_t size,  
                        int (*compare)(const void*, const void*));
```

Problem 4 [40 points]

In this question we get a stack of ints. The implementation is given in a separate file. You should not make assumptions about the exact implementation details.

You may only use the following functions to access the stack.

```
typedef struct {  
    // not known  
} stack_int_t;  
// creates a new stack  
stack_int_t* stack_create();  
// pushes a given item to the stack  
void stack_push(stack_int_t* s, int item);  
// removes the top element from the stack and returns it  
// Precondition: stack is not empty  
int stack_pop(stack_int_t* s);  
// checks if the stack is empty  
bool stack_is_empty(stack_int_t* s);  
// frees the stack  
void stack_free(stack_int_t* s);
```

The zip file contains the object file lib/stack_int.o, which implements the functions. This .o file is specific for the CSIL machines.

If you want to run your code on your machine, you will need to create an object file stack_int.o for your operating system.

This is done using the command “gcc -c stack_int.c” in the lib folder.

- a) [10 pts] Write a function that gets a stack of ints and returns the number of elements in it.

When the function returns, the stack must be in its initial state.

```
// returns the size of the stack  
int stack_size(stack_int_t* s)
```

- b) [15 pts] Write a function that gets a stack of ints and a predicate pred, and returns the number of elements in the stack satisfying pred.

When the function returns, the stack must be in its initial state.

```
// counts the number of x's in the stack with pred(x)=true  
int stack_count(stack_int_t* s, bool (*pred)(int))
```

- c) [15 pts] Write a function that gets a stack of ints and reverses the order of the elements in the stack.

```
// reverses the order of the elements in the stack  
void stack_reverse(stack_int_t* s)
```

**** Remember: you should only use the provided interface, and not assume that stack_t is implemented in a certain way. For grading your solution this implementation will change.**