

CMPT125, Spring 2023

Homework Assignment 3

Due date: Sunday, March 26, 2023, 23:59

You need to solve the first two problems in ***assignment3.c***.
For the third problem, you need to declare the struct in ***set_t.h***,
and solve it in ***set_t.c***.

Submit all three files to CourSys.

Solve all 3 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.

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 solve the first two problems in ***assignment3.c***.
2. Problem 3 needs to be solved in ***set_t.h*** and ***set_t.c***.
3. If necessary, you may add helper functions.
4. You should not add `main()`, because it will interfere with `main()` in the test file.
5. Submit only the files ***assignment3.c***, ***set_t.h***, ***set_t.c*** to CourSys.

Problem 1 [20 points].

Let $p > 2$ be a parameter, and define the following variant of the Fibonacci sequence:

- $\text{fib_p}(0) = 1, \text{fib_p}(1) = 1$
- $\text{fib_p}(n) = (\text{fib_p}(n-1) + \text{fib_p}(n-2)) \pmod p$ for all $n \geq 2$

For example, for $p=7$, the sequence is [1, 1, 2, 3, 5, 1, 6, 0, 6, 6, 5, 4, 2, 6...]

```
int fib_p(unsigned int n, unsigned int p);
```

Your function needs to return the correct answer for all $n < 1,000,000$ within 1 second.

Problem 2 [30 points]

Write the following four functions:

a) The function gets an array A of length n of ints, and a boolean predicate pred. It returns the first (smallest) index i such that $\text{pred}(A[i]) == \text{true}$. If no such element is not found, the function returns -1.

```
int find(int* A, int n, bool (*pred)(int));
```

b) The function gets an array A of length n of ints, and a boolean predicate pred. It returns the number of indices i such that $\text{pred}(A[i]) == \text{true}$.

```
int count(int* A, int n, bool (*pred)(int));
```

c) The function gets an array A of length n of ints, and a function f. It applies f to each element of A.

```
void map(int* A, int n, int (*f)(int));
```

d) The function gets an array A of length n of ints, and a function f. The function f gets 2 ints and works as follows:

1. Start with accumulator = A[0]
2. For $i=1 \dots \text{length}-1$ compute $\text{accumulator} = f(\text{accumulator}, A[i])$
3. Return accumulator

For example, if f computes the sum of the two inputs, then reduce() will compute the sum of the entire array.

```
int reduce(int* A, int n, int (*f)(int,int));
```

Problem 3 [50 points] - ****solve the problem in set_t.h and set_t.c****

Define in **set_t.h** the struct `set_t` representing a set of ints, i.e., an unordered collection of ints without duplicates. You may assume that the size of the set will always be at most 1,000.

You will need to implement the following functions in **set_t.c**:

- `set_t* set_create_empty();`
 - The function returns a pointer to an empty set.
- `int set_size(set_t* A);`
 - The function gets a pointer to the set A, and returns the size of the set.
- `void set_insert(set_t* A, int x);`
 - The function gets a pointer to the set A and a number x, and adds x to the set. If x was already in the set, the function has no effect on A.
- `void set_remove(set_t* A, int x);`
 - The function gets a pointer to the set A and a number x. The function removes x from the set. Assumption: x belongs to A.
- `bool set_contains(set_t* A, int x);`
 - The function gets a pointer to the set A and a number x. The function returns true if x belongs to A, and returns false otherwise.
- `int set_map(set_t* A, int (*f)(int));`
 - The function gets a set A and a function f, and applies f to all elements of A. Note that after applying f some values might become duplicates, in which case you need to remove the duplicates. The function returns the size of the new set A.
- `void set_free(set_t* A);`
 - The function gets a pointer to the set A, and frees the memory.

For example:

```
set_t* A = create_empty_set(); // A = {}
for (int i=10; i<15; i++)
    set_insert(A, i);
// here A = {10,11,12,13,14}
set_map(A, plus1); // A becomes {11,12,13,14,15}
set_insert(A, 9); // A becomes {9,11,12,13,14,15}
set_remove(A, 13); // A becomes {9,11,12,14,15}
set_insert(A, 1); // A becomes {1,9,11,12,14,15}
set_insert(A, 12); // A doesn't change, 12 was already in A
set_map(A, mod5); // A becomes {0,1,2,4}
set_free(A);
```

Here `mod5` is defined as: `int mod5(int x) {return x%5;}`

and `plus1` is defined as: `int plus1(int x) { return x+1;}`

* You need to decide how to represent the set using the struct in `set_t.h`. You may add any data fields you think are needed. The test cases will test that your functions work with each other as specified. The test cases will not check the implementation details.