Home Assignment 1

Project template

Now you have [template (https://github.com/cubazis/inno_ansic_spring/tree/dev/assignments/ha-1/template)](https://github.com/cubazis/inno_ansic_spring/tree/dev/assignments/ha-1/template) for your first home assignment.

Project Architecture

```
src/task.h
src/task.c
src/binarize.h
src/binarize.c
tests/test_task.c
tests/test_binarize.c
```

Tasks

That's what's important! If you suddenly use some macros in the tests, then write them in test_task.c itself or in test_binarize.c. If you use macros in the sources - then write them in the header of the sources. But do not store source code macros in tests and vice versa test macros in source code. Keep clean!

I recommend familiarizing yourself with the basic concepts of the language by solving the problems from Kernighan Ritchie. And only then proceed to the binarizer.

Binarizator

Note that the declarations of *binarize_u* and *binarize_s* functions have changed. Now they must return *char**.
That was done to make them testable.

Please finish implementation of these two functions and support them by two test cases. Test cases stubs are already in test_binarize.c.

Example of binarization

```
3 >> 00000000 00000011
111 >> 00000000 01101111
32767 >> 01111111 11111111
32768 >> 10000000 00000000
65535 >> 11111111 11111111
65536 >> 00000000 00000001 00000000 00000000
4294967296 >> 00000000 00000000 00000000 00000001 00000000 00000000 00000000
00000000
```

KR Tasks

Не отклоняйтесь от предлагаемой архитектуры, потому как и я и вы будете тестить исходники ваших товарищей. Проще всего не отклоняться от данного шаблона чтобы просто скопировать тестируемый исходник.

Разбейтесь на команды по 4. Составы впишите в файлик (скину ссыль). Каждая команда заводит репозиторий и проект с указанной архитектурой папок и файлов. Вы решаете

1. задачу бинаризатора дописывая код в месте где это необходимо и покрывая более обширными тестами. Причем в шаблоне что я вам дал бинарищатор просто печатает в консоль доп код поданного числа. Переделайте код так чтобы функции binarize_u и binarize_s не печатали в консоль а возвращали массив соответствующей длины. Чтобы это можно было покрыть тестами. Причем давайте так. Если вы подали на вход число 3 то вы знаете что функция вернёт два бита и массив в который будет записан результат создаёте длинной в 2 бита. По указателю ничего не передаете, предполагаем что вы это не знаете.
   В test_bibarize.c пишите тесты на все случаи жизни.

2. задачи от кернигана.
   Все функции пишите в task.c
   Все тесты к ним в test_task.c

Формат тестов такой же как в шаблоне cmake_first_step. Отдельный тест кейс тестирует одну задачу. Но внутри тест стюта должно быть избыточное на ваш взгляд покрытие проверкам . Обратите внимание на то что поверка на точность расчетов должна учитывать что разные архитектуры и разные ОСи (даже внутри одного семейства ОС) могут давать разные результаты при мат подсчётах. Ошибка копится медленно но тем не менее.

Таким образом для проверки работоспособности программ достаточно брать папки src и tests. И все это получается системно независимым. Поэтому ребята с разными ОС все равно могут проверять программы друг друга на корректность.

После того как ваш репозиторий заполнен вы добавляете меня в коллабораторы и ставите задачу (задачи git) с формулировкой "мы все"

По задачам.
[1.20] вы пишете функцию detab. И покрываете ее тестами. На вход подаются массивы символов. На выход массивы символов. Функция должна заменять табуляции на 4 пробела.

```
char* detab(const char input[]);
```

[1.21] функция entab. Заменяет пробелы на табуляции и пробелы. Например 5 пробелов --> 1 таб + 1 пробел, 10 --> 2 Таба + 2 пробела.

```
char* entab(const char input[]);
```

[1.22] пишете функцию enter. функция принимает первым параметром n. Вторым параметром принимает массив символов содержащие условную строку. Если строка длиннее чем n, то после каждой группы n символов вставляется символ '\n'. Таким образом входной текст трансформируется в текст с переносами на следующую строку после каждых n символов

```
char* enter(int n, const char input[]);
```

[1.23] функция flush принимает массив символов содержащий программу на C. Удаляет из него все комментарии, то есть олнострочные // и многострочные /* /* */. Обратите внимание что бывают вот такие комментарии.

/* Бла

- Бла
- Бла
  */

Функция возвращает массив символов без комментариев. Критерий проверки - компилируемость очищенного от комментариев текста программы

```
char* flush(const char input[]);
```

[1.24] не делать.

По задачам.
[2.1] решение есть в файле лаб сессии. Эта задача для самообучения и она делается не в проекте и только для себя.

[2.2] аналргично для самоподготовки.

[2.3] а вот это уже в проект и
покрыть тестами. Функция htoi

```
int htoi(const char s[]);
```

[2.4] функция squeeze

```
char* squeeze(const char s1[], const char s2[]);
```

[2.5] функция any

```
int any(const char s1[], const char s2[]);
```

[2.6] функция setbits

```
unsigned setbits(unsigned x, int p, int n, unsigned y);
```

[3.1] функция binsearch. Читайте задание внимательно. Там требуется переписать код.

```
int binsearch(int x, int v[], int n);
```

[3.2] функция escape. Обратную к ней писать не надо

```
char* escape(const char from[]);
```

[3.3] функция expand

```
char* expand(const char s1[]);

```

[3.4] функция itoa. Читайте задание внимательно. Требуется доработать код.

```
char* itoa(int n);
```

[3.5] функция itob. Читайте задание внимательно

```
char* itob(int n, int b);
```

Задача 3.6 не нужна

[4.1] функция strrindex.

```
int strrindex(const char s[], const  char t[]);
```

[4.2] функция atofe. Читайте внимательно. Доработать для экспоненциальной записи

```
double atofe(const char s[]);
```

`

4.3 для саморазвития. В task.c она не нужна, и в test_task соответственно. Калькулятор там для обратной польской записи.

Deadline

After **12.05 28/05** you commits will not be checked

Extra Stuff

The array_changer function just to demonstrate how to work with returning of array from function

- C language doesn't have strings
- array in a C language is just a segment of memory
- C lang doesn't have ability to pass array as parameter in a function
- we suppose that you don't know about memory allocation through pointers

So, we provide you simple demo function which change 'c' char in input array

- input parameter: constant array of chars
- output: pointer to memory segment in which resulting array starts
- extra stuff: we provide you three define macros in task.h.

You can use them in your solutions
if you don't want to use them please, don't delete them - somebody may use it
please look for test case for array_changer function in test_task.c

```
char* array_changer(const char c[])
{
    char i = 0, j = 0;
    STRING_LEN(i, c);
    char* b = ALLOCATE(i);

    for (;j < i;) {
        if(b[j] == 'c')
            b[j] = 'b';
        else
            b[j] = c[j];
        j++;
    }
    return b;
}
```

And tcase for array_changer function

```
START_TEST (test_array_changer)
{
    const char input[] = "abcba";
    const char pattern[] = "abbba";

    /** change 'c' to 'b' */

    char* output = array_changer(input);

    /** initialise k because ck_assert need's it to be sure
     *  that he compare something initialised with 1
     *  because if COMPARATOR will not change it value, k will not have any
value
     *  */
    int k = 1;

    /** get k, k, output, pattern
     *  compare output array with pattern array by elements
     *  return 0 if they are not equal
     * */
    COMPARATOR(k, output, pattern);

    /** check statement is true */
    ck_assert(1 == k);

}
END_TEST
```