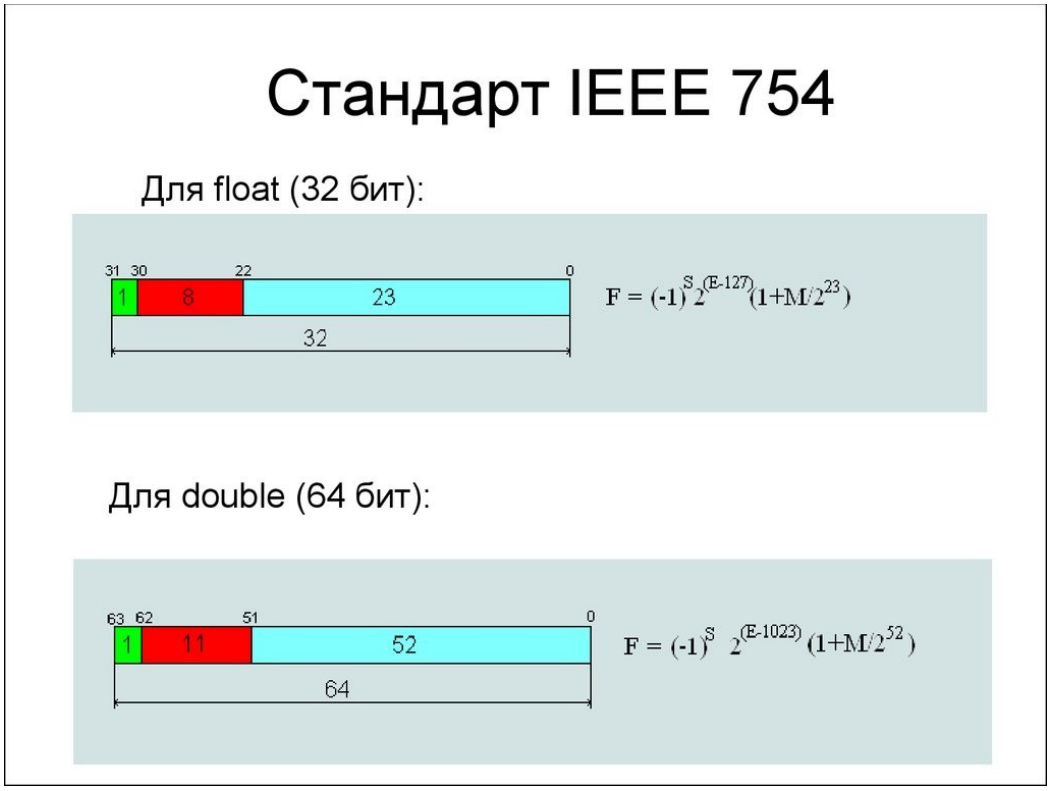
1. Написать пару функций, одна из которых будет устанавливать все биты, а другая — снимать по маске. Простейшая вещь, но довольно полезная (особенно в контексте того, что именно так работает большинство флешек).

2. Прочитать из потока ввода число типа float и затем вывести его на экран в формате a*2\^b, где a и b — десятичные числа, причём a в диапазоне от [0,1; 1).



3. Реализовать работу с битовым массивом, то есть функции

int setbit(int index, int bit, uint32_t array[])

int getbit(int index, uint32_t array[])

такие, чтобы устанавливали (забирали) index-й бит по счёту в массиве array.

Тоже несложно, но мозг прочистит.

4. Написать функцию set_value, устанавливающую значение по заданному в виде целого числа адресу.

5. Написать typedef для 32-битного типа с названием my_type таким образом, чтобы можно было обращаться к байтам(поле с названием byte), полусловам(u16), словам (u32) и числам с плавающей запятой (f).

6. Написать функцию print_array_similarly_to_intel_hex(void* addr, int len) (название условное, естественно), выводящую на экран массив в следующем текстовом формате (все числа выводятся шестнадцатеричными числами в верхнем регистре, по два символа на байт):

1. Первая строка: двоеточие, затем все байты адреса, кроме двух младших;
2. Каждая следующая строка содержит данные, разложенные по полям:
    1. Двоеточие (':')
    2. Первое поле — два байта (младшие два байта тех данных, которые выводятсяв данной строке — для первой строки данных это будут просто два младших байта переменной addr, для каждой следующей строки прибавляем к предыдущему адресу число байтов данных, записанных в предыдущей строке).
    3. Второе поле — число 00
    4. Третье поле — данные (не более 16 байтов, подряд)
    5. Четвёртое поле — один байт, считающийся как сумма всех предыдущих байтов в строке (с переполнением, разумеется).

    Пример использования:

```
#include <stdio.h>
#include <stdint.h>
uint8_t array[100] =
{0,1,2,3,4,5,6,7,8,9,0xA,0xB,0xC,0xD,0xE,0xF,0x10,0x11};
int main()
{
 print_array_similar_to_intel_hex(array, 20);

 return 0;
}
:000040
:40400000010203040506070809 0A0B0C0D0E0FF8
:405000 10110000B1
```

Это посложнее, но для тренирвоки перед курсовиком… Ну, в идеале: сделать парсер файла (функцию с прототипом `int parse_xpm(char ** xpm, uint16_t *result)`) формата XPM (с одной

стороны, PBM проще, с другой — есть две его версии, что не очень удобно в учебных целях). Задача: есть ~~текстовый файл~~ массив строк заданного формата с xpm внутри (один символ на точку, два цвета), необходимо найти строку, определяющую чёрный цвет и проанализировать файл, записывая в массив форму кривой — то есть её координату Y для каждого шага по X (считаем, что эта зависимость однозначна). Возвращаемое значение функции — ширина изображения.