Задача 1

Вам задаётся объём памяти, который надо уметь адресовать на некоторой конкретной аппаратной платформе. Нужно вычислить, сколько занимает указатель на данной платформе.

Например, если нужно адресовать 1 Гб памяти, то указатель должен уметь хранить значения от 0 до 1073741823. Для хранения всех значений из этого диапазона потребуется 30 бит. Так как память выделяется побайтово, а в байте 8 бит, то реально указатель будет занимать в памяти 4 байта.

Что стоит запомнить из этой задачи: (1) размер указателя связан с конкретной аппаратной платформой, (2) организация работы с указателями связана с максимально доступной программе оперативной памятью.

Формат входных данных

Вводится ровно одно целое число - объём памяти **в мегабайтах**, которую надо суметь адресовать.

Формат выходных данных

Количество **байт**, которого минимально достаточно для хранения адресов.

Примеры

Pf	Вывод
16	3
17	4
16384	5

Задача 2

Вам задано описание структуры. Вот такое:

```
struct something {
   int a;
   bool b;
   char c;
};
```

Вам не известно, как компилятор на конкретной платформе решит расположить структуру в памяти. (Это вообще очень небанальный вопрос, связанный с кучей нюансов работы оборудования и выходящий за рамки текущего курса C++.)
Есть массив из таких структур. Напишите функцию, которая принимает на вход адрес первой структуры в массиве и номер требуемой структуры из массива, после чего вычисляет и возвращает адрес этой требуемой структуры. Прототип функции:

```
struct something* calc_address(struct something* start, unsigned int number);
```

Что стоит запомнить из этой задачи: (1) адреса можно и нужно вычислять в ходе выполнения программы, (2) арифметика указателей - великая вещь.

Задача 3

Есть функция вот с таким прототипом:

```
bool do_some_work(int* a, int* b);
```

Вам неизвестно, что именно она делает. Но что-то делает. (Для отладки можете написать себе любую её реализацию.)

Считайте со стандартного потока ввода два числа a и b, передайте их в функцию. Если она вернула true, выведите на экран разность (a - b) после работы функции. Если она вернула false, выведите на экран сумму (a + b) после работы функции.

Что стоит запомнить из этой задачи: (1) иногда функции принимают на вход указатели, получая тем самым доступ к самим переменным, а не к их копиям, (2) функция может и что-то сделать с переданными по указателю параметрами, и вернуть какое-то значение.

Задача 4

Напишите функцию, которая меняет местами две переданные переменные. Менять нужно именно значения у вызвавшего, не локальные копии. Прототип функции:

```
void swap(int* a, int* b);
```

Что стоит запомнить из этой задачи: ещё раз о том, что параметры могут быть переданы по значению (копии) или по указателю (адрес у вызвавшего).

Задача 5

Напишите функцию, которая принимает на вход массив, выделяет блок памяти под его копию, копирует исходный массив, возвращает указатель на сформированную копию. Прототип функции:

```
int* copyarr(int* a, unsigned int size);
```

Первый аргумент - указатель на начало массива для копирования, второй параметр - количество элементов в исходном массиве.

В данном случае освобождение выделенного блока памяти остаётся, разумеется, на вызвывшем функцию.

Что стоит запомнить из этой задачи: блок динамической памяти можно выделить в одной точке программы и передать в другую.

Задача 6

Напишите функцию, которая транспонирует матрицу, переданную на вход. Прототип функции:

```
int** transpose(int** matrix, unsigned int N, unsigned int M);
```

Матрица хранится в следующем виде: массив указателей на строки в количестве N штук, в каждой строке M элементов.

Нужно создать новую матрицу и вернуть её. Портить исходную матрицу не нужно. (Не забудьте, что матрица не квадратная, так что её размеры поменяются местами при транспонировании.)

Что стоит запомнить из этой задачи: указатели на указатели - это вариант нормы жизни.