ACC: Накопитель

Привычные нам (императивные) подходы к написанию программ сейчас многое заимствуют из функционального программирования. В области обработки данных таким популярным шаблоном проектирования является тройка функций высшего порядка Map / Filter / Reduce, обрабатывающих массивы при помощи вспомогательных функций (элементарных операций).

- **Map** (отобразить), также *transform* (преобразовать), *apply* (применить) — преобразует массив в другой массив такой же длины, применяя к каждому элементу заданную функцию и сохраняя результат.
- **Filter** (отбросить), также *remove* (убрать) — преобразует массив в другой массив, возможно, меньшего размера, применяя к каждому элементу заданную функцию и решая, оставить или отбросить его.
- **Reduce** (уменьшить), также *accumulate* (накопить), *fold* (свернуть) — преобразует массив в одно значение, последовательно применяя заданную функцию к элементам.

Чтобы использовать этот подход в Си и аналогичных языках, требуется уметь передавать функцию в качестве параметра алгоритма. В Си для этого предназначены указатели на функцию.

В этой задаче требуется реализовать механизм **накопителя** из стандартной библиотеки Си++ (accumulate = reduce) при обработке массива. Простейшее применение — нахождение суммы элементов массива, но в зависимости от вспомогательной функции меняется и результат. Основная функция имеет вид:

```
int Accumulate(int const a[], int n, int s0, binary_op_t* op);
```

На вход она принимает массив `a` размером `n`, и начальное значение накапливаемой величины `s0`. Последним аргументом передается указатель на функцию `op`, некоторую бинарную операцию:

```
typedef int binary_op_t(int sk, int ak);
```

Задача функции — начиная со стартового значения `s0` последовательно обойти массив, вызывая бинарную операцию `op` и передавая в качестве первого аргумента переменную-накопитель, а в качестве второго — очередной элемент массива. Возвращаемое операцией `op` значение будет являться новым значением накопителя. В итоге функция `Accumulate` возвращает значение накопителя после обработки всего массива.

$$s_n = ((s_0 \odot a_0) \odot a_1) \odot \ldots \odot a_{n-1},$$

или

$$s_{k+1} = s_k \odot a_k, \quad k = 0 \ldots (n-1).$$

По шагам:

0. **Подготовьте массив (0 баллов).** Заполните массив случайными числами от -10 до 10. Напечатайте его на экране. Для этого напишите функции `Fill` и `Print`.

1. **Реализуйте аккумулятор (+1 балл).** Напишите саму функцию `Accumulate`, реализующую описанный выше алгоритм. Вызовите её, передавая по очереди три разных бинарных операции, для того, чтобы в итоге получить:

   - сумму элементов массива,
   - максимальное значение в массиве,
   - количество отрицательных элементов.

   Не забывайте делать программу дружелюбной, печатайте поясняющие сообщения, а не просто числа.

   ```
   int BinarySum(int sk, int ak);
   int BinaryMax(int sk, int ak);
   int BinaryNeg(int sk, int ak);
   ```

2. **Избавьтесь от дублирования кода (+1 балл).** Работу с самими функциями `BinaryXXX()` организуйте таким образом, чтобы избежать дублирования кода и copy-paste, т.е. храните их в массиве (структур с описанием и указателем на функцию) и организуйте по ним цикл.

3. (*) **Изучите ещё одно применение (+1 бонус).** Измените заполнение массива так, чтобы его элементы оказались равными 0 или 1. Теперь интерпретируйте массив как запись числа в двоичном виде (старший бит лежит в нулевом элементе массива), и создайте вспомогательную функцию `BinaryBin`, которая позволит собрать это число из двоичной записи. **Подсказка:** используйте схему Горнера как алгоритм перевода между системами счисления.