

**Министерство науки и высшего образования Российской
Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО**

Факультет безопасности информационных технологий

Дисциплина:

«Алгоритмы и структуры данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«Алгоритм цифровой сортировки методом LSD

через кольцевые очереди на базе связного списка»

Выполнил:

Ганшин Ярослав Андреевич, студент группы N3250

(подпись)

Проверил:

Ерофеев Сергей Анатольевич

(отметка о выполнении)

(подпись)

Санкт-Петербург
2025 г.

1. ВВЕДЕНИЕ

1.1.Цель работы

Разработать на языке C программу цифровой сортировки методом LSD, считывающую целые числа из текстового файла и реализующую распределение по корзинам через **кольцевые очереди на базе связного списка**, с возможностью выбора между статическим и динамическим хранением данных.

1.2.Алгоритм цифровой сортировки методом LSD через кольцевые очереди на базе связного списка

- **Организация структуры очереди.**Каждый элемент очереди представляет структура

Node,содержащая значение и указатель на следующий узел.Для управления очередью используется один указатель на «хвост» кольца, когда очередь пуста, он равен NULL.

- **Подготовка корзин.**Для цифр от 0 до 9 создаётся массив из десяти пустых

очередей.Каждая очередь инициализируется особым образом. Первый узел в очереди ссылается сам на себя. Образуется замкнутое кольцо.

- **Чтение и разбор входных данных.**Пользователь вводит путь к текстовому файлу и

выбирает режим хранения (статический или динамический массив).Все числа в файле считываются строка за строкой, пробелы и запятые нормализуются в единичные пробелы, а экспоненциальная нотация конвертируется в целые значения.

- **Разделение на отрицательные и неотрицательные.**

Извлечённые числа разбиваются на две группы:

- 1) **отрицательные**, для которых сохраняется их абсолютное значение;
- 2) **неотрицательные**, которые остаются без изменений.

- **Определение числа разрядов.**В каждой группе находится наибольшее по модулю

значение M.Число проходов по разрядам d определяется количеством цифр в десятичном представлении M.

- **Основной цикл поразрядной сортировки.**Повторять d раз для каждой группы отдельно:

- **Распределение.**Для каждого элемента x вычисляется его текущая цифра: $\text{digit} = (x / \text{exp}) \% 10$.

Элемент добавляется в соответствующую кольцевую очередь методом enqueue — операция вставки нового узла в хвост списка.

- **Сборка.**Проходя по всем десяти очередям от 0 до 9, из каждой методом dequeue

последовательно извлекаются все элементы и записываются обратно в массив.После завершения очереди всех корзин параметр exp умножается на 10, цикл повторяется для следующего разряда.Для отрицательных чисел после всех проходов дополнительно выполняется обратный порядок,возвращается знак “-”.

- **Формирование итогового результата.**Отрицательные и неотрицательные значения

объединяются в один массив.Результат выводится на экран в виде упорядоченной последовательности.

1.3 Пример работы алгоритма с кольцевыми очередями на базе связного списка

Рассмотрим входной массив: [170, 45, -75, 90, -802, 24, 2, -66].

1. Разделение на группы

- Положительные: [170, 90, 45, 24, 2].
- Отрицательные (модули): [75, 802, 66].

2. Определение числа разрядов для группы положительных:

Максимальное значение = 170 \rightarrow 3 разряда \rightarrow $\text{exp} = 1, 10, 100$.

3. Иллюстрация работы корзины для $\text{digit} = 0$ (единицы) :

Инициализация:

Создаем bucket0: $\text{tail} = \text{NULL}$, $\text{size} = 0$.

Первый элемент (170):

$\text{digit} = 0$.

Создается первый узел с $\text{data} = 170$.

Так как $\text{tail} = \text{NULL}$, новый узел указывает сам на себя, tail принимает адрес этого узла, size становится 1.

Второй элемент (90):

$\text{digit} = 0$.

Создается новый узел (new) с $\text{data} = 90$.

Поскольку $\text{tail} \neq \text{NULL}$, вставляем сразу после tail :

$\text{new} \rightarrow \text{next} = \text{tail} \rightarrow \text{next}$ (голова очереди с 170),

$\text{tail} \rightarrow \text{next} = \text{new}$,

$\text{tail} = \text{new}$,

size увеличивается до 2.

Состояние после вставок:

$\text{bucket0.size} = 2$,

$\text{bucket0.tail} \rightarrow \text{data} = 90$,

$\text{bucket0.tail} \rightarrow \text{next} \rightarrow \text{data} = 170$,

(образуется замкнутое кольцо: $170 \rightarrow 90 \rightarrow 170$).

Удаление (dequeue):

Первый dequeue:

head = tail->next (170), возвращаем 170,

tail->next сдвигается на следующий узел (90), size = 1.

Второй dequeue:

head = tail->next (90), возвращаем 90,

так как head == tail, сбрасываем tail = NULL, size = 0.

Очередь пуста.

4.Распределение и сборка по разрядам

exp = 1 (единицы)

Положительные: 170 и 90 в bucket0; 2 в bucket2; 24 в bucket4; 45 в bucket5.

Сборка → [170, 90, 2, 24, 45].

Отрицательные: 75 в bucket5; 802 в bucket2; 66 в bucket6.

Сборка → [802, 75, 66].

exp = 10 (десятки)

Положительные распределены в bucket0,2,4,7,9 → сборка → [2, 24, 45, 170, 90].

Отрицательные распределены в bucket0,6,7 → сборка → [802, 66, 75].

exp = 100 (сотни)

Положительные: bucket0 и bucket1 → сборка → [2, 24, 45, 90, 170].

Отрицательные: bucket0 и bucket8 → сборка → [66, 75, 802].

5.Коррекция отрицательных

Разворачиваем [66, 75, 802] и возвращаем знак → [-802, -75, -66].

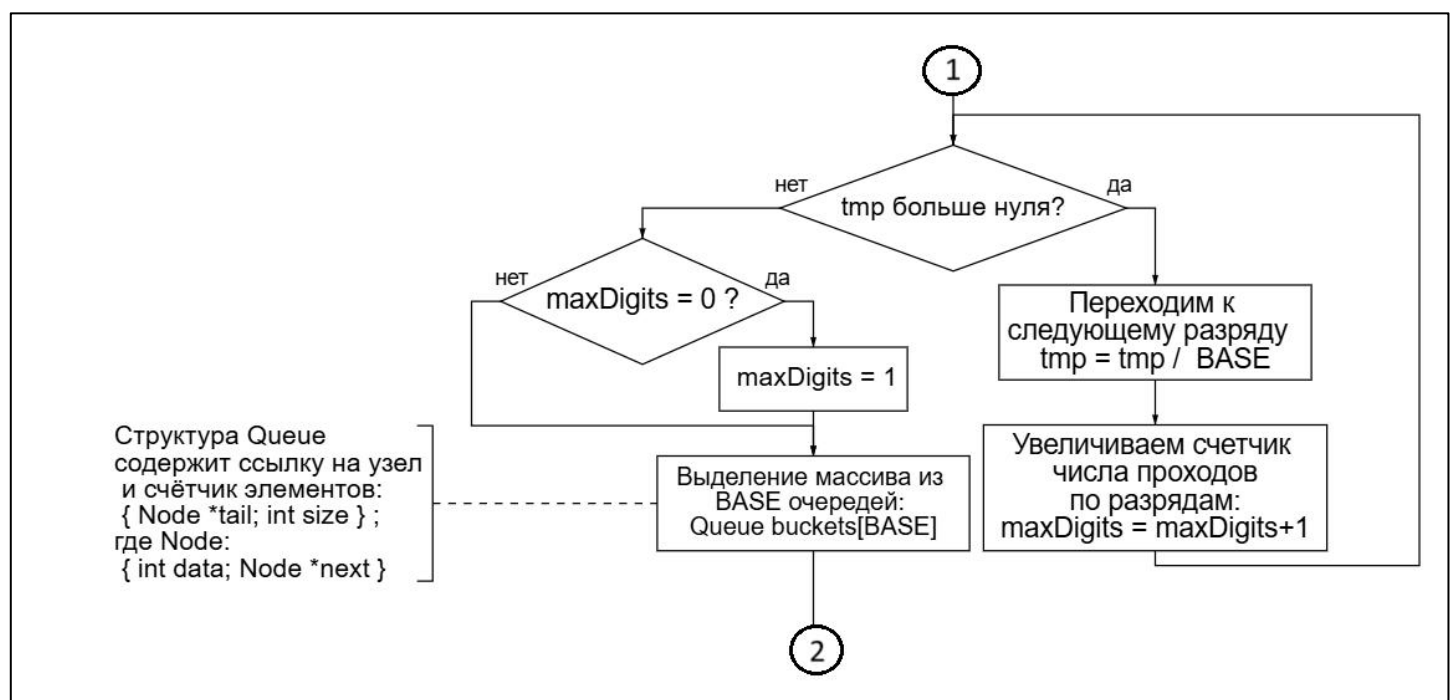
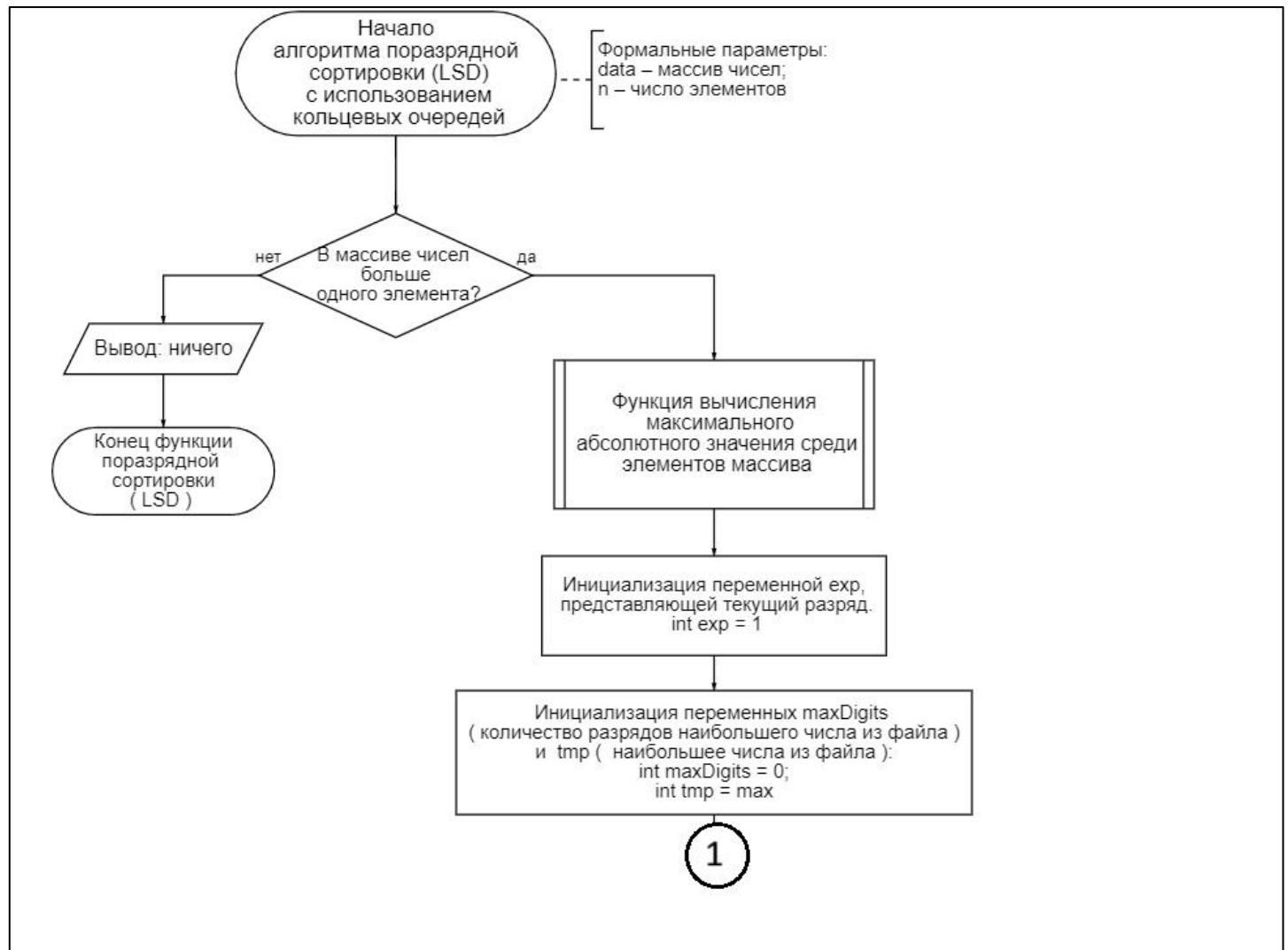
6.Формирование итогового массива

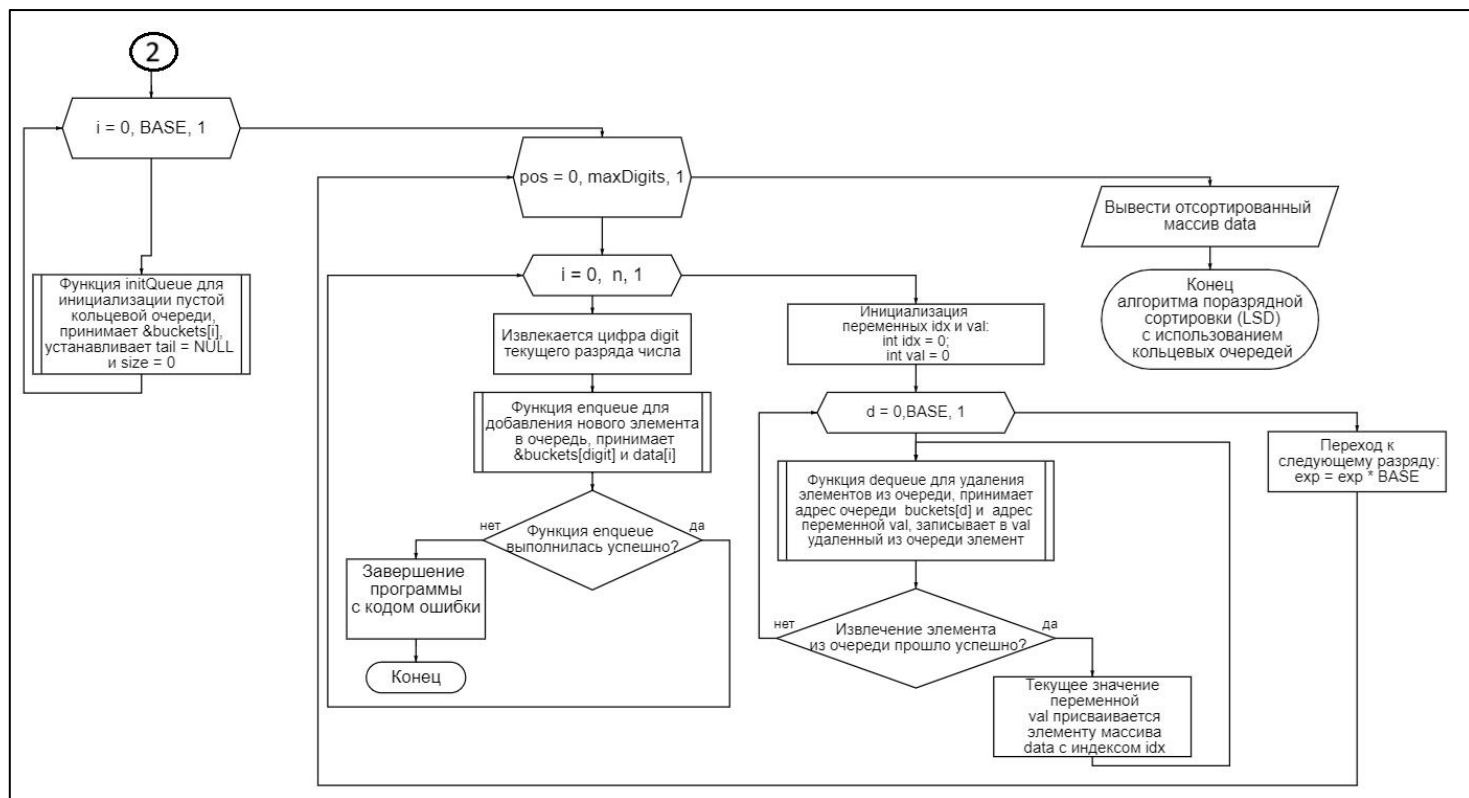
Объединяем отрицательные и положительные: [-802, -75, -66, 2, 24, 45, 90, 170].

2.ТАБЛИЦА ИСПОЛЬЗУЕМЫХ ПЕРЕМЕННЫХ

| Имя | Тип | Диапазон | Емкость (байт) | Роль в программе |
|-----------------|-----------------------|---|-----------------|---|
| input_file | FILE * | - | 8 | Указатель на входной файл |
| filename | char[256] | $[-2^7; 2^7-1]$ | 256 | Буфер для имени файла |
| buffer | char[MAX_LINE_LENGTH] | $[-2^7; 2^7-1]$ | MAX_LINE_LENGTH | Буфер для чтения строки |
| storage_type | int | {0, 1, 2} | 4 | Выбор типа массива |
| item_count | int | $[0; 2^{31} - 1]$ | 4 | Количество прочитанных чисел |
| capacity | int | $[0; 2^{31} - 1]$ | 4 | Начальная ёмкость динамического массива |
| static_data | int[STATIC_SIZE] | $[0; 2^{31} - 1]$ | STATIC_SIZE * 4 | Статический массив |
| dynamic_data | int * | - | 8 | Указатель на динамический массив |
| token | char * | - | 8 | Токен при разбиении строки |
| num | int | $[-2^{31}; 2^{31} - 1]$ | 4 | Преобразованное число |
| negative_values | int * | - | 8 | Массив для отрицательных значений (abs) |
| positive_values | int * | - | 8 | Массив для неотрицательных значений |
| neg_cnt | int | $[0; 2^{31} - 1]$ | 4 | Число отрицательных элементов |
| pos_cnt | int | $[0; 2^{31} - 1]$ | 4 | Число неотрицательных элементов |
| max | int | $[0; 2^{31} - 1]$ | 4 | Максимальное абсолютное значение |
| exp | int | $[0; 2^{31} - 1]$ | 4 | Текущий разряд (1, 10, 100...) |
| idx | int | $[0; 2^{31} - 1]$ | 4 | Индекс при сборке |
| value | long / double | $[-2^{63}; 2^{63} - 1] / [-1.7 \cdot 10^{308}; 1.7 \cdot 10^{308}]$ | 8 | Переменная при strtol / при strtod |
| maxDigits | int | $[0; 2^{31} - 1]$ | 4 | Число проходов по разрядам |
| tmp | int | $[0; 2^{31} - 1]$ | 4 | Вспомогательная переменная при вычислении разрядов |
| pos | int | $[0; 2^{31} - 1]$ | 4 | Индекс текущего разряда в цикле сортировки |
| val | int | $[0; 2^{31} - 1]$ | 4 | Переменная для хранения извлеченного из очереди значения |
| tail | Node * | - | 8 | Указатель на конец структуры Queue |
| next | Node * | - | 8 | Указатель на следующий узел |
| head | Node * | - | 8 | Головной узел |
| size | int | $[0; 2^{31} - 1]$ | 4 | Количество элементов в Queue |
| data | int | $[0; 2^{31} - 1]$ | 4 | Значение узла |
| pValue | int * | - | 8 | Указатель на значение, которое дотаем из Queue |
| digit | int | $[0; 2^{31} - 1]$ | 4 | Часть числа, рассматриваемого во время сортировки по корзинам |
| n | Node * | - | 8 | Новый узел |

3.БЛОК-СХЕМЫ





Функция `initQueue` для
инициализации пустой
кольцевой очереди

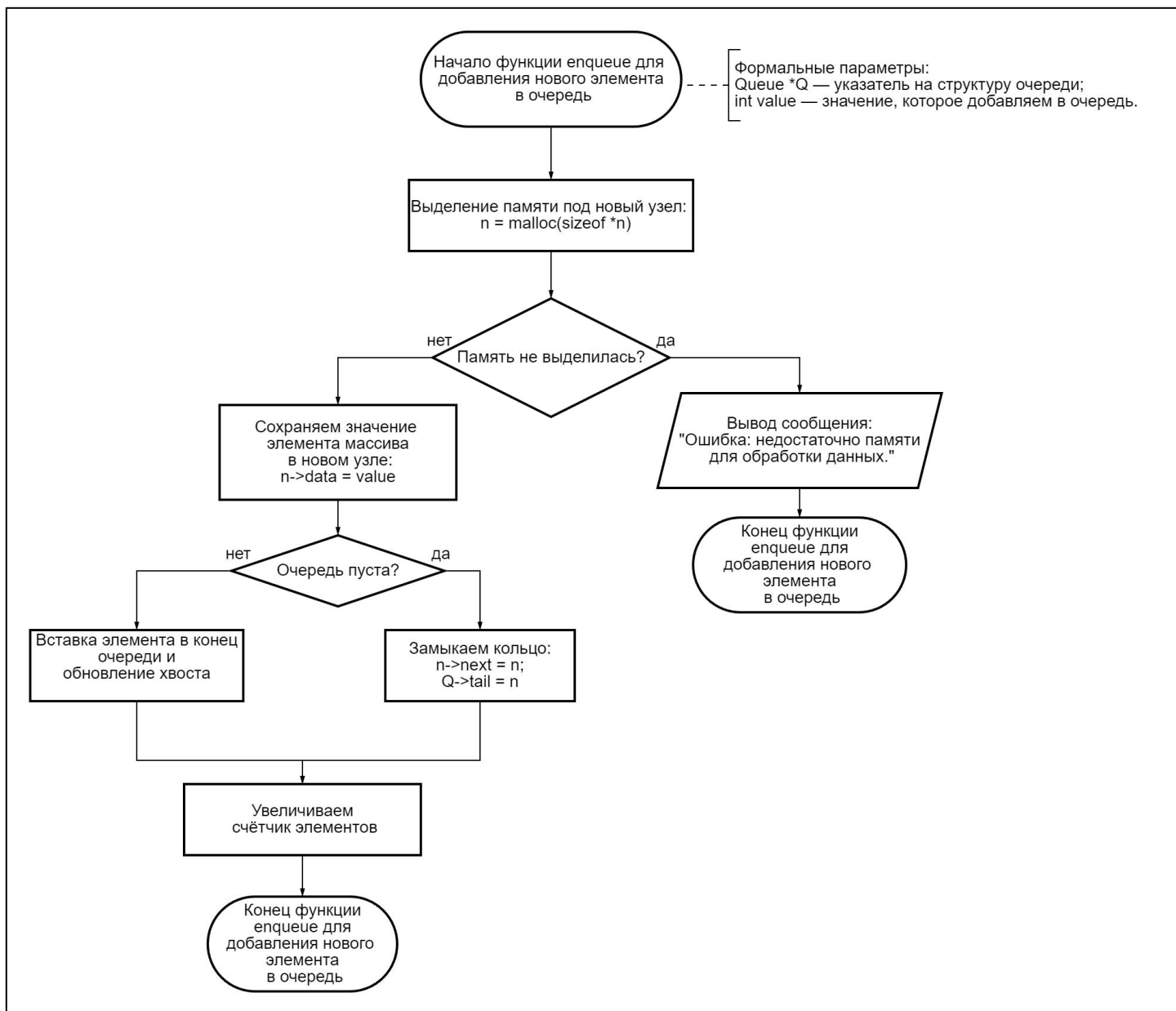
--- [Формальный параметр:
`Queue *Q` — указатель на структуру очереди

Устанавливаем указатель
`tail` в `NULL`

----- [Очередь пуста
(нет узлов)

Присваиваем полю `size`
структуры `Q` значения 0

Конец функции
`initQueue` для
инициализации пустой
кольцевой очереди



Начало функции dequeue
для удаления
элементов из очереди

Формальные параметры:
Queue *Q — указатель на очередь;
int *pValue — указатель на возвращаемое значение

нет да
Очередь пуста?

Определяем головной
узел head

Присваиваем значение
головного узла
через указателю
pValue

Передаем в функцию
поразрядной сортировки
информацию, что
очередь пуста

Конец функции dequeue
для удаления
элементов из очереди

нет да
В очереди только
один узел?

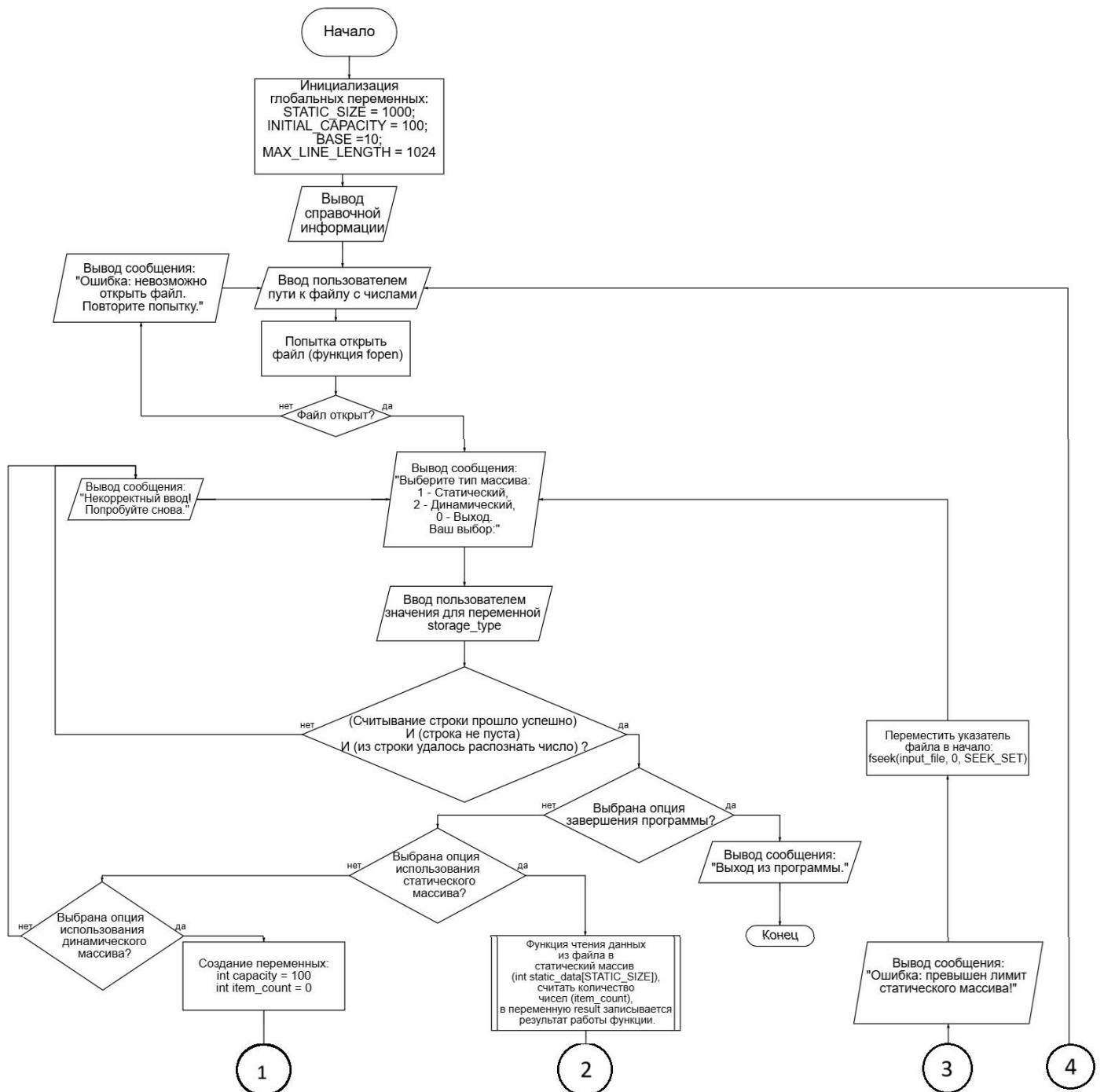
Переподключить хвост
к следующему узлу,
пропуская головной

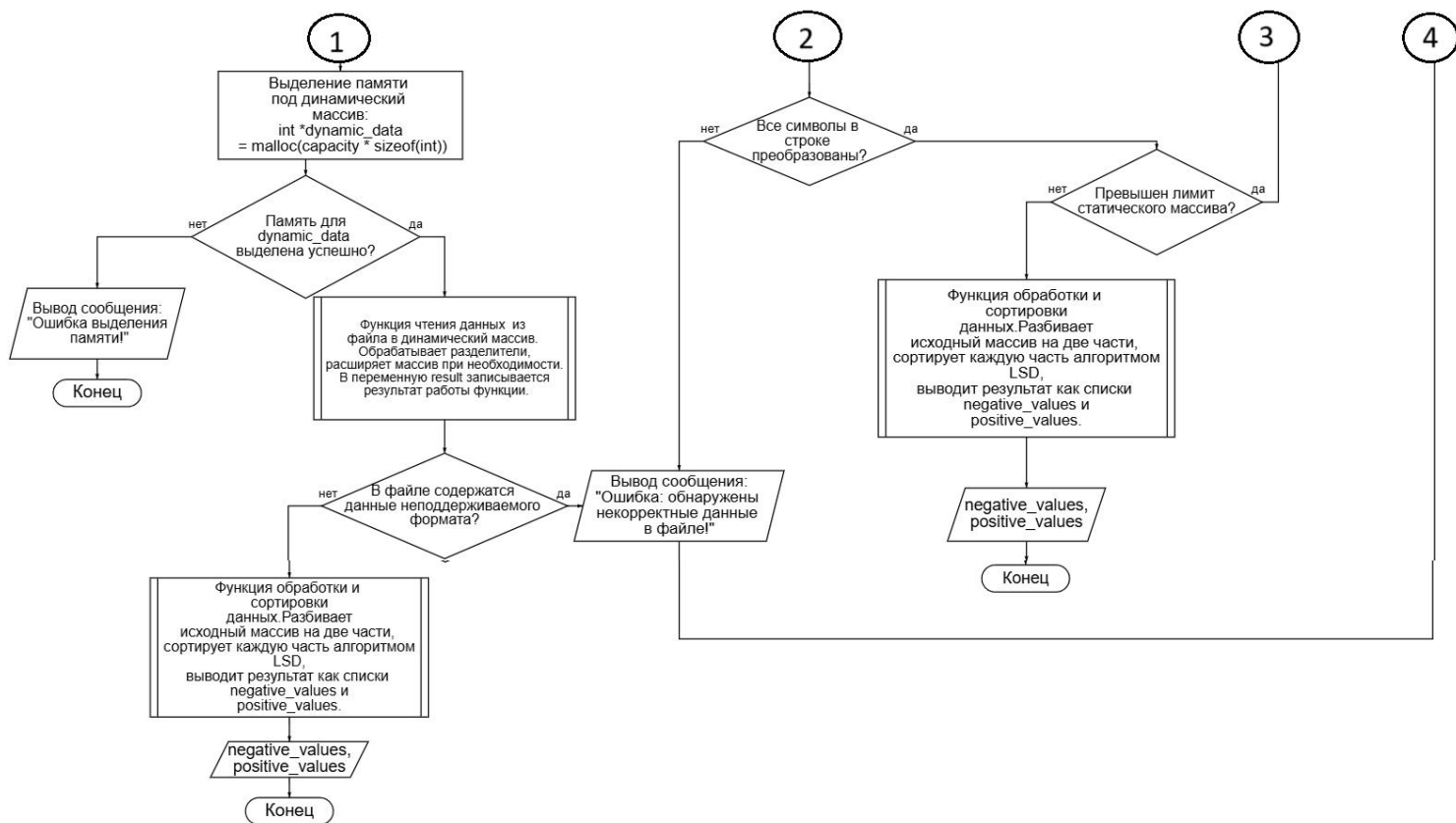
Очередь становится
пустой

Освобождаем
память узла head

Уменьшаем счётчик
элементов

Конец функции dequeue
для удаления
элементов из очереди





Формальные параметры:
*file – файл;
*data – массив чисел;
max_size – размер массива;
*count – счётчик элементов

Начало функции чтения
данных из файла в
статический массив

Инициализировать
строковый буфер
buffer размером
MAX_LINE_LENGTH

Попытка чтения
строки из файла
(вызов fgets)

Получена ли строка
из файла?

нет

да

Возврат
значения 0

Конец
функции чтения данных
из файла в
статический массив

Функция нормализации
разделителей в строке.
Берет buffer.
Заменяет
символы-разделители
на пробелы

Разделение строки
на токены с
помощью
strtok(buffer, " ")

Есть ли токен?

нет

да

Функция преобразования токена
в число. Берет token, проверяет на
наличие экспоненты, приводит
к десятичному виду

Весь токен
успешно
преобразован
в число?

нет

да

Возврат
значения -1

Конец
функции чтения данных
из файла в
статический массив

Достигнут ли
максимальный
размер массива?

нет

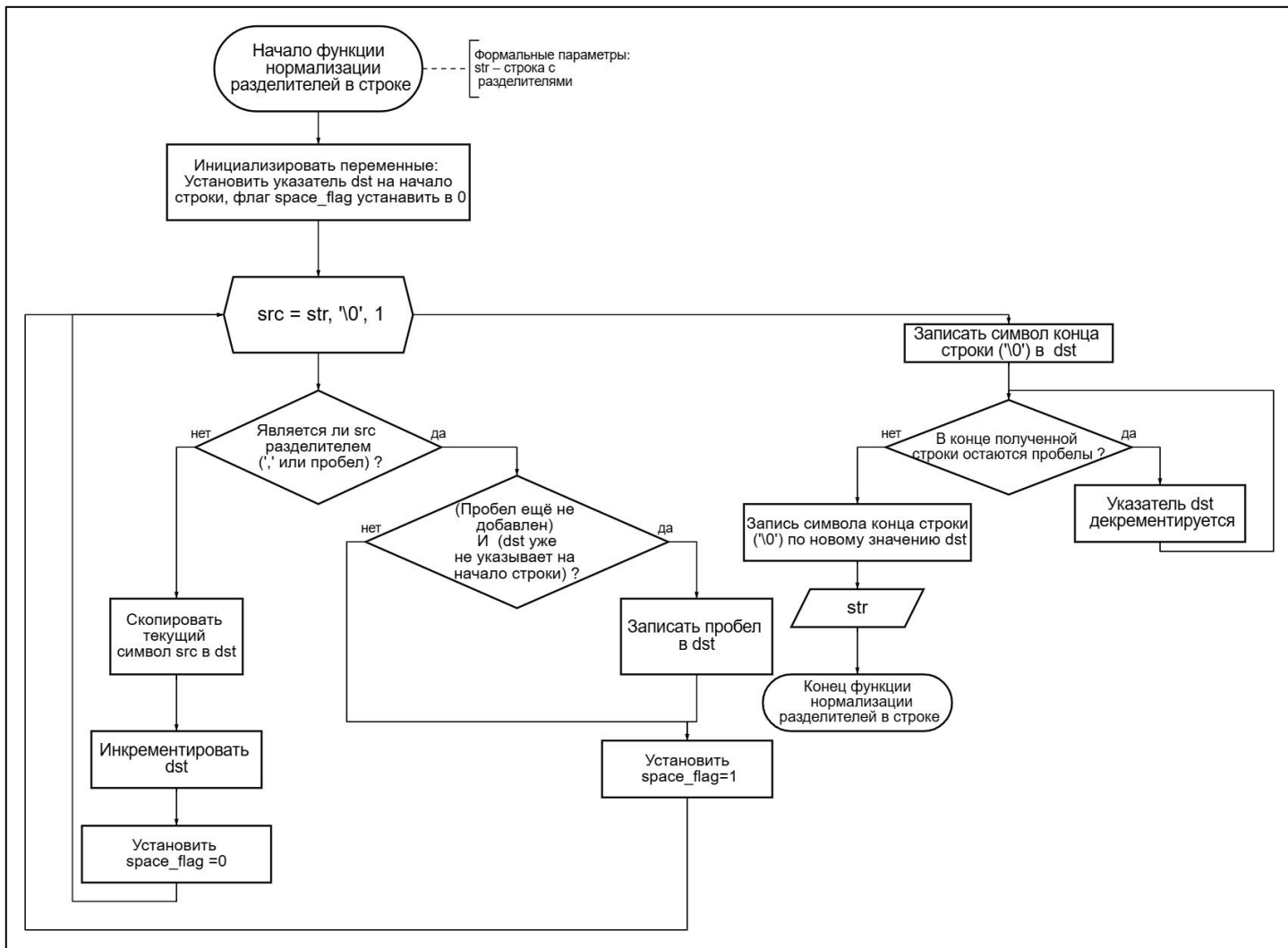
да

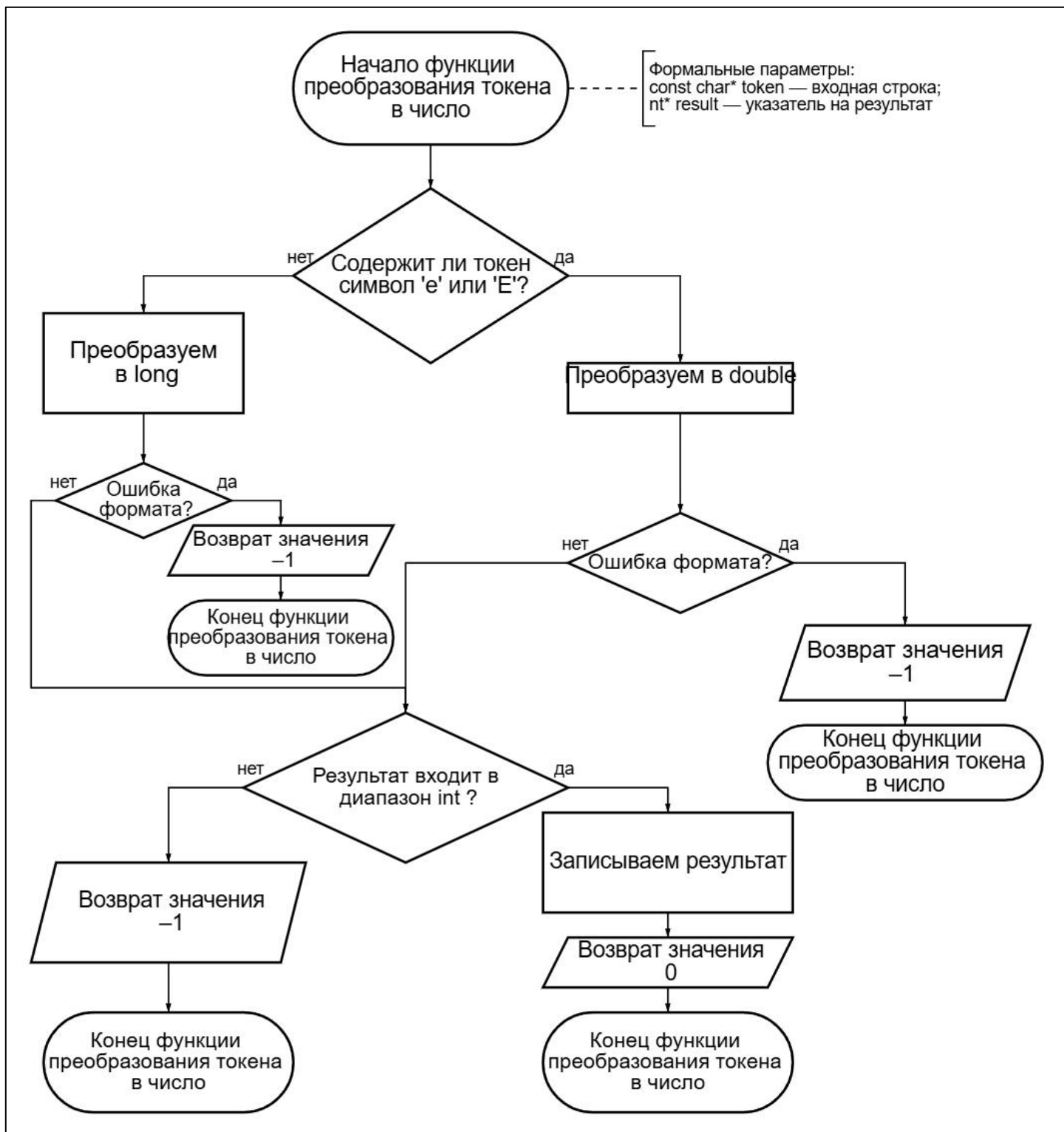
Возврат
значения -2

Конец
функции чтения данных
из файла в
статический массив

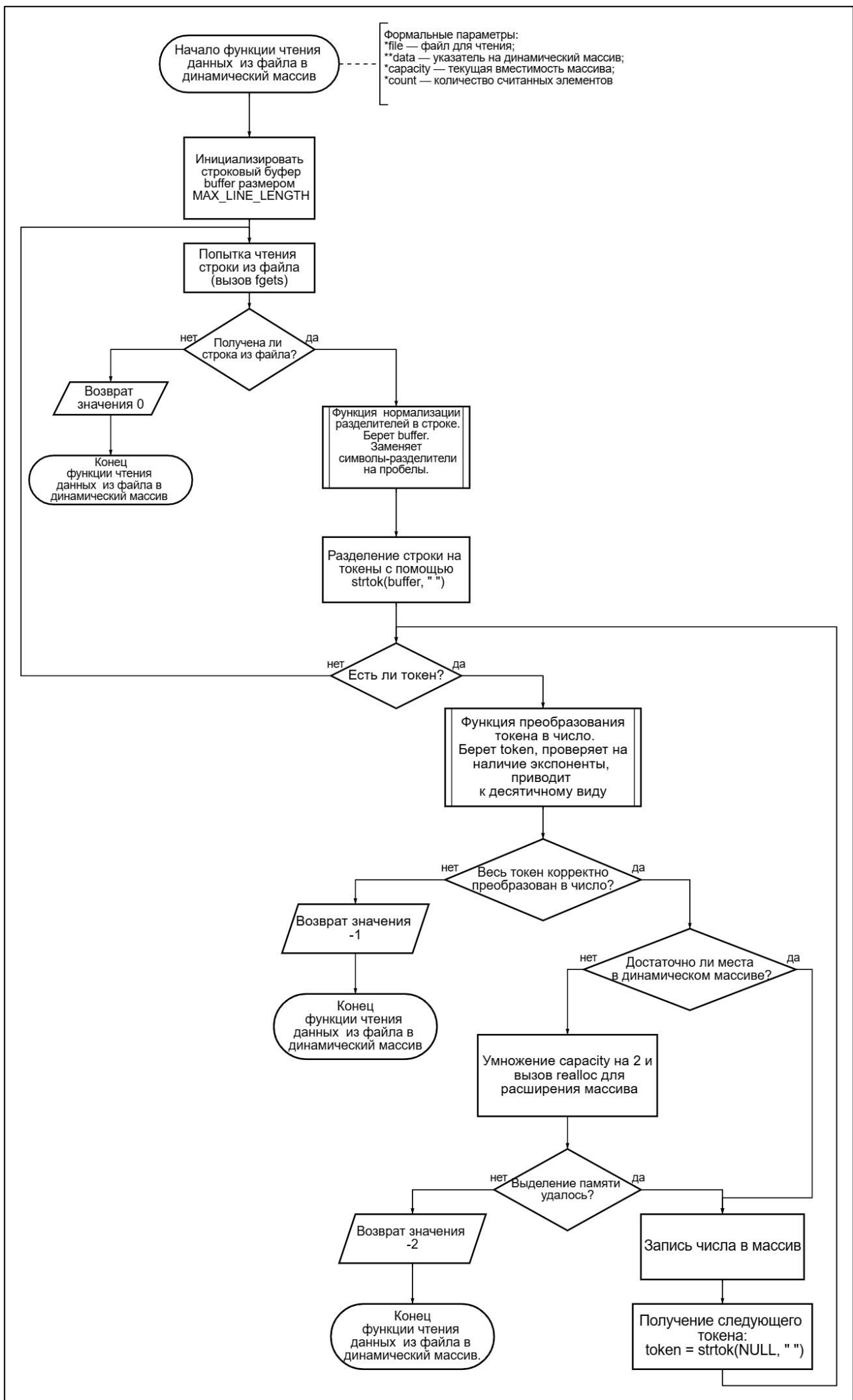
Сохранение числа в
массиве и увеличение
счётчика count

Получение
следующего
токена с помощью
strtok(NULL, " ")





Формальные параметры:
*file — файл для чтения;
**data — указатель на динамический массив;
*capacity — текущая вместимость массива;
*count — количество считанных элементов



Начало функции вычисления
максимального
абсолютного значения среди
элементов массива

Формальные параметры:
data – массив чисел;
n – число элементов

Инициализация переменной max
абсолютным значением
первого элемента массива

i = 1, n, 1

Вывести max

Вычисление
абсолютного значения
текущего элемента
массива с индексом i

Конец функции вычисления
максимального
абсолютного значения среди
элементов массива

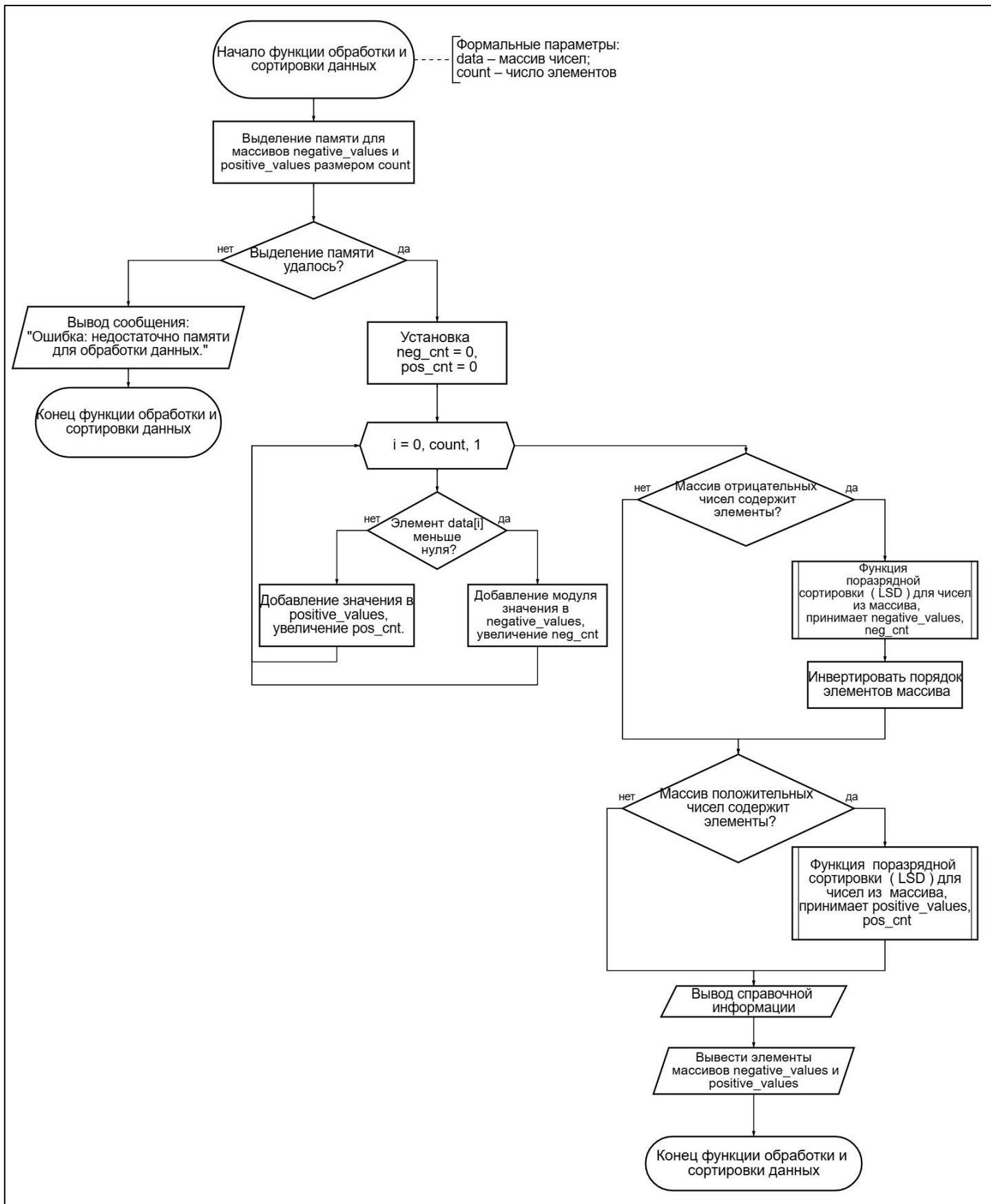
Текущее абсолютное
значение
больше максимального
значения max?

нет

да

Присвоить max = current





4. КОД ПРОГРАММЫ С КОММЕНТАРИЯМИ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <stdbool.h>

#define STATIC_SIZE    1000 // макс. размер статического массива
#define INITIAL_CAPACITY 100 // начальная ёмкость динамического массива
#define BASE          10 // основание системы счисления (число корзин)
#define MAX_LINE_LENGTH 1024 // макс. длина считываемой строки

// Прототипы функций
int getMaxAbsolute(int arr[], int n); // макс. элемент
void lsd_sort(int data[], int n); // LSD-сортировка массива
void processDelimiters(char* str); // нормализация разделителей
int readStaticData(FILE* file, int data[], int max_size, int* count); // в статич. массив
int readDynamicData(FILE* file, int** data, int* capacity, int* count); // в динамич. массив
void processAndDisplay(int data[], int count); // разделение, сортировка, вывод
int convertExponentialToInt(const char* token, int* result); // в десятичный вид

// кольцевая очередь на базе связанного списка

typedef struct Node {
    int data; // значение узла
    struct Node *next;
} Node;

typedef struct {
    Node *tail; // NULL, если пустая
    int size; // текущее число элементов
} Queue;

// инициализация пустой очереди
static void initQueue(Queue *Q) {
    Q->tail = NULL;
    Q->size = 0;
}

// добавление в конец
static bool enqueue(Queue *Q, int value) {
    Node *n = malloc(sizeof *n); // память под новый узел
    if (!n) { // если не удалось выделить
        printf("Ошибка: недостаточно памяти для обработки данных.\n"); // текст ошибки
        return false;
    }
    n->data = value;
    if (Q->tail == NULL) { // если очередь пуста
        n->next = n; // сам на себя
        Q->tail = n;
    } else { // если есть хотя бы один узел
        n->next = Q->tail->next; // следующий узел после нового — головной
        Q->tail->next = n; // старый хвост указывает на новый узел
    }
}
```

```

    Q->tail = n; // хвост на новый узел
}
Q->size++;
return true;
}

```

// удаление

```

static bool dequeue(Queue *Q, int *pValue) {
    if (Q->tail == NULL) return false; // головной узел
    Node *head = Q->tail->next; // головной узел
    *pValue = head->data;
    if (head == Q->tail) { // если в очереди только один узел
        Q->tail = NULL;
    } else {
        Q->tail->next = head->next; // пропускаем головной узел
    }
    free(head);
    Q->size--;
    return true;
}

```

// преобразование строки token в число int

```

int convertExponentialToInt(const char* token, int* result) {
    if (strchr(token, 'e') || strchr(token, 'E')) {
        char* end;
        double value = strtod(token, &end);
        if (*end != '\0') return -1;
        if (value > INT_MAX || value < INT_MIN) return -1; // проверяем диапазон int
        *result = (int)value;
        return 0;
    } else {
        char* end;
        long value = strtol(token, &end, 10);
        if (*end != '\0') return -1;
        if (value > INT_MAX || value < INT_MIN) return -1; // проверяем диапазон int
        *result = (int)value;
        return 0;
    }
}

```

// поиск макс. абсолютного значения

```

int getMaxAbsolute(int data[], int n) {
    int max = data[0];
    for (int i = 1; i < n; i++) {
        int current = data[i];
        if (current > max)
            max = current;
    }
    return max;
}

```

//LSD-сортировка через кольцевые очереди

```

void lsd_sort(int data[], int n) {
    if (n <= 1) return; // мало элементов

    int max = getMaxAbsolute(data, n);
    int exp = 1;

```

// число разрядов

```

int maxDigits = 0;
{
    int tmp = max;
    while (tmp > 0) {
        tmp /= BASE;
        maxDigits++;
    }
    if (maxDigits == 0) maxDigits = 1;
}

// 10 корзин — очередей
Queue buckets[BASE];
for (int i = 0; i < BASE; i++)
    initQueue(&buckets[i]);
// по каждому разряду
for (int pos = 0; pos < maxDigits; pos++) {
    // распределение
    for (int i = 0; i < n; i++) {
        int digit = (data[i] / exp) % BASE;
        if (!enqueue(&buckets[digit], data[i])) {
            // в enqueue вывели ошибку
            exit(1);
        }
    }
    // сборка
    int idx = 0;
    int val = 0;
    for (int d = 0; d < BASE; d++) {
        while (dequeue(&buckets[d], &val)) {
            data[idx++] = val;
        }
    }
    exp *= BASE;
}

}

// приведение разных разделителей к одиночному пробелу

void processDelimiters(char* str) {
    char* dst = str; // куда пишем очищенный символ
    int space_flag = 0; // предыдущий записанный символ был пробелом?
    for (char* src = str; *src; src++) { // проходим по всем символам исходной строки
        if (*src == ' ' || isspace((unsigned char)*src)) { // если разделитель или пробельный символ
            if (!space_flag && dst > str)
                *dst++ = ' '; // копируем оригинальный символ
            space_flag = 1; // устанавливаем флаг пробела
        } else {
            *dst++ = *src;
            space_flag = 0;
        }
    }
    *dst = '\0';
    while (dst > str && isspace((unsigned char)*(dst - 1))) dst--;
    *dst = '\0';
}

//чтение в статический массив

int readStaticData(FILE* file, int data[], int max_size, int* count) {
    char buffer[MAX_LINE_LENGTH];
    while (fgets(buffer, sizeof(buffer), file)) {
        processDelimiters(buffer); // очистка разделителей
        char* token = strtok(buffer, " ");
    }
}

```

```

while (token) {
    int num;
    if (convertExponentialToInt(token, &num) != 0)
        return -1; // ошибка формата
    if (*count >= max_size) // проверка вместимости
        return -2; // превышен лимит
    data[(*count)++] = num;
    token = strtok(NULL, " ");
}
}
return 0;
}

```

// чтение в динамический массив

```

int readDynamicData(FILE* file, int** data, int* capacity, int* count) {
    char buffer[MAX_LINE_LENGTH];
    while (fgets(buffer, sizeof(buffer), file)) {
        processDelimiters(buffer); // очистка разделителей
        char* token = strtok(buffer, " ");
        while (token) {
            int num;
            if (convertExponentialToInt(token, &num) != 0)
                return -1; // формат неверен
            if (*count >= *capacity) {
                *capacity *= 2;
                int* new_data = realloc(*data, *capacity * sizeof(int));

                *data = new_data;
            }
            (*data)[(*count)++] = num;
            token = strtok(NULL, " ");
        }
    }
    return 0;
}

```

// разделение, сортировка и вывод

```

void processAndDisplay(int data[], int count) {
    int* negative_values = malloc(count * sizeof(int)); // для отрицательных
    int* positive_values = malloc(count * sizeof(int)); // для положительных
    if (!negative_values || !positive_values) {
        printf("Ошибка: недостаточно памяти для обработки данных.\n");
        free(negative_values);
        free(positive_values);
        return;
    }
    int neg_cnt = 0, pos_cnt = 0;
    for (int i = 0; i < count; i++) {
        if (data[i] < 0)
            negative_values[neg_cnt++] = abs(data[i]);
        else
            positive_values[pos_cnt++] = data[i];
    }
    if (neg_cnt > 0) { // сортируем модули отрицательных и переворачиваем порядок
        lsd_sort(negative_values, neg_cnt);
        for (int i = 0; i < neg_cnt / 2; i++) {
            int tmp = negative_values[i];
            negative_values[i] = negative_values[neg_cnt - 1 - i];
            negative_values[neg_cnt - 1 - i] = tmp;
        }
    }
    if (pos_cnt > 0) // сортируем положительные

```

```

    lsd_sort(positive_values, pos_cnt);

    printf("\nРезультат сортировки:\n"); // вывод результата
    for (int i = 0; i < neg_cnt; i++)
        printf("%d ", -negative_values[i]);
    for (int i = 0; i < pos_cnt; i++)
        printf("%d ", positive_values[i]);
    printf("\n\n");

    free(negative_values);
    free(positive_values);
}

int main() {

    // описание работы программы
    printf("Код принимает данные из текстового файла в десятичном формате.\n");
    printf("Числа могут разделяться пробелами или запятыми.\n");

    FILE* input_file = NULL; // указатель на файл
    char filename[256]; // буфер для пути
    int storage_type; // выбор типа массива

get_file:
    while (1) { // ввод и открытие файла
        printf("Укажите путь к файлу: ");
        if (!fgets(filename, sizeof(filename), stdin)) {
            if (feof(stdin)) return 0; // выход при EOF
        }
        filename[strcspn(filename, "\n")] = '\0';
        input_file = fopen(filename, "r");
        if (!input_file) {
            printf("Ошибка: невозможно открыть файл.Повторите попытку.\n");
            continue; // повторить запрос
        }
        break;
    }

choose_storage:
    while (1) {
        char per[128]; // выбор типа хранения
        printf("\nВыберите тип массива:\n1 - Статический\n2 - Динамический\n0 - Выход\nВаш выбор: ");
        char* fret = fgets(per, sizeof(per), stdin);
        if (!fret && feof(stdin)) return 0;
        if (sscanf(per, "%d", &storage_type) != 1 || storage_type < 0 || storage_type > 2) {
            printf("Некорректный ввод!Попробуйте снова.\n");
            continue;
        }
        if (storage_type == 0) {
            printf("Выход из программы.\n");
            fclose(input_file);
            return 0;
        }
        break;
    }

    if (storage_type == 1) {
        int static_data[STATIC_SIZE]; // статический буфер
        int item_count = 0;
        int result = 0;
        result = readStaticData(input_file, static_data, STATIC_SIZE, &item_count);
        if (result == -1) {
            printf("Ошибка: обнаружены некорректные данные в файле!\n");
            fclose(input_file);
        }
    }
}

```

```

    goto get_file; // повтор открытия
}
if (result == -2) {
    printf("Ошибка: Превышен лимит статического массива!\n");
    fseek(input_file, 0, SEEK_SET); // сброс к началу
    goto choose_storage;
}
processAndDisplay(static_data, item_count); // сортировка и вывод

} else {
    int capacity = INITIAL_CAPACITY;
    int item_count = 0;
    int* dynamic_data = malloc(capacity * sizeof(int)); // динамическая память
    if (!dynamic_data) {
        printf("Ошибка: недостаточно памяти для обработки данных.\n");
        fclose(input_file);
        return 1;
    }
    int result = readDynamicData(input_file, &dynamic_data, &capacity, &item_count);
    if (result == -1) {
        printf("Ошибка: обнаружены некорректные данные в файле!\n");
        free(dynamic_data);
        fclose(input_file);
        goto get_file;
    }
    processAndDisplay(dynamic_data, item_count); // сортировка и вывод
    free(dynamic_data);
}
// закрытие файла и завершение
fclose(input_file);
return 0;
}

```


5. ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМА

```
// 6
// инициализация пустой очереди
static void initQueue(Queue *Q) {
    Q->tail = NULL; // 3
    Q->size = 0; // 3
}

//в худшем случае будет 33
// добавление в конец
static bool enqueue(Queue *Q, int value) {
    Node *n = (Node*) ( (Node *) (Тшвумalloc(sizeof(Node)))); // 4
    if (n == NULL) { // 1
        printf("Ошибка: недостаточно памяти для обработки данных.\n"); // 1
        return false; // 1
    }
    n->data = value; // 3
    if (Q->tail == NULL) { // 3
        n->next = n; // 3
        Q->tail = n; // 3
    } else {
        n->next = Q->tail->next; // 7
        Q->tail->next = n; // 5
        Q->tail = n; // 3
    }
    Q->size++; // 6
    return true; // 1
}

//в худшем случае будет 30
// удаление
static bool dequeue(Queue *Q, int *pValue) {
    if (Q->tail == NULL) // 3
        return false; // 1
    Node *head = Q->tail->next; // 5
    *pValue = head->data; // 4
    if (head == Q->tail) { // 3
        Q->tail = NULL; // 3
    } else {
        Q->tail->next = head->next; // 7
    }
    free(head); // 1
    Q->size--; // 6
    return true; // 1
}

//в худшем случае будет  $2 + 1 + 1 * n + 2 * (n - 1) + ((n - 1) * (2 + 1 + 1)) + 1$ 
//в обычном варианте будет  $2 + 1 + 1 * n + 2 * (n - 1) + (n - 1) * (2 + 1) + 1 * kol + 1$ 
// поиск макс. абсолютного значения
int getMaxAbsolute(int data[], int n) {
    int max = data[0]; // 2
    for (int i = 1; i < n; i++) {
        // в худшем случае будет  $1 + 1 * n + 2 * (n - 1) + ((n - 1) * (2 + 1 + 1))$ 
        //в обычном варианте будет  $1 + 1 * n + 2 * (n - 1) + (n - 1) * (2 + 1) + 1 * kol$ ,
        //где kol - кол-во чисел, стоящих в порядке очереди, где последующее больше или
        // равно предыдущему
        int current = data[i]; // 2
        if (current >= max) // 1
            max = current; // 1
    }
}
```

```

    }
    return max; //1
}

```

//LSD-сортировка через кольцевые очереди

```

void lsd_sort(int data[], int n) {

```

```

    if (n <= 1) // 1
        return; // 1

```

```

    int max = getMaxAbsolute(data, n);

```

```

    // в худшем случае  $2 + 1 + 1 * n + 2 * (n - 1) + ((n - 1) * (2 + 1 + 1)) + 1$ 

```

```

    int exp = 1; // 1

```

```

    // число разрядов

```

```

    int maxDigits = 0; // 1

```

```

    {
        int tmp = max; // 1
        while (tmp > 0) { //log_BASE(max)* 4
            tmp /= BASE; // 2
            maxDigits++; // 2
        }
        if (maxDigits == 0) // 1
            maxDigits = 1; // 1
    }

```

```

    // корзины

```

```

    Queue buckets[BASE]; // 1

```

```

    for (int i = 0; i < BASE; i++)

```

```

    //для for

```

```

    //  $1 + 1 * (BASE + 1) + 2 * BASE + BASE * (6 * BASE)$ 
    initQueue(&buckets[i]); // 6

```

```

    for (int pos = 0; pos < maxDigits; pos++) {

```

```

        //для for

```

```

        //в худшем случае будет  $1 + 1 * (maxDigits + 1) + 2 * maxDigits + (1 + 1 * (n + 1) + 2 *$ 

```

```

         $/* n + (24 * 1 + (n - 1) * 33 + 1 * n + 4 * n) * maxDigits + 1 * maxDigits + 1 *$ 

```

```

         $/* maxDigits + (1 + 1 * (BASE + 1) + 2 * BASE + (34n - 4 + 5(BASE - 1)) *$ 

```

```

         $/* BASE) * maxDigits + 2 * maxDigits$ 

```

```

        //в обычном случае будет  $1 + 1 * (maxDigits + 1) + 2 * maxDigits + (1 + 1 * (n + 1) + 2 *$ 

```

```

         $/* n + (1 * n + 24 * un + (n - un) * 33 + 4 * n) * maxDigits + 1 * maxDigits + 1 *$ 

```

```

         $/* maxDigits + (1 + 1 * (BASE + 1) + 2 * BASE + (34n - 4m + 5(BASE - m)) *$ 

```

```

         $/* BASE) * maxDigits + 2 * maxDigits$ 

```

```

        // распределение

```

```

        for (int i = 0; i < n; i++) {

```

```

            //для for

```

```

            //в худшем случае будет  $1 + 1 * (n + 1) + 2 * n + (24 * 1 + (n - 1) * 33 + 1 * n + 4 *$ 

```

```

             $/* n)$ 

```

```

            // в обычном случае будет  $1 + 1 * (n + 1) + 2 * n + (1 * n + 24 * un + (n - un) *$ 

```

```

             $/* 33 + 4 * n)$ , где un - уникальные корзины ( уникальные значения digit )

```

```

            int digit = (data[i] / exp) % BASE; // 4

```

```

            if (enqueue(&buckets[digit], data[i]) == false) {

```

```

                //для enqueue

```

```

                //в лучшем случае будет  $24 + 1$ 

```

```

                //в худшем случае будет  $33 + 1$ 

```

```

                exit(1); //1
            }

```

```

        }

```

```

    //сборка

```

```

    int idx = 0; //1

```

```

int val = 0; // 1
for (int d = 0; d < BASE; d++) {
    //в обычном варианте
    //  $1 + 1 * (BASE + 1) + 2 * BASE + (34n - 4m + 5(BASE - m)) * BASE$ 
    //в худшем случае
    //  $1 + 1 * (BASE + 1) + 2 * BASE + (34n - 4 + 5(BASE - 1)) * BASE$ 
    while (dequeue(&buckets[d], &val)) {
        //для dequeue
        //в худшем случае будет  $30 + 1$ 
        //иначе будет  $26 + 1$ 
        //если пуста, то будет  $4 + 1$ 
        //для while
        //в обычном варианте:
        //  $34n - 4m + 5(BASE - m)$ , где m - полные, n - элементов во всех, всего BASE //корзин
        //в худшем случае (одна корзина):
        //  $34n - 4 + 5(BASE - 1)$ 
        data[idx++] = val; // 3
    }
}
exp *= BASE; // 2
}
}

```

5.1. Анализ зависимости времени и количества операций от объёма входных данных

0 элементов в считываемом массиве.

```
Укажите путь к файлу: data_0_worst_case.txt

Выберите тип:
1
2
0
Your Answer: 2

0.000001 seconds
0 operation
0 input data
```

4012 элементов в считываемом массиве.

```
Укажите путь к файлу: data_4000_worst_case.txt

Выберите тип:
1
2
0
Your Answer: 2

0.001356 seconds
1480844 operation(s)
4012 input data
```

8023 элемента в считываемом массиве.

```
Укажите путь к файлу: data_8000_worst_case.txt

Выберите тип:
1
2
0
Your Answer: 2

0.002805 seconds
2960881 operation(s)
8023 input data
```

12035 элементов в считываемом массиве.

```
Укажите путь к файлу: data_12000_worst_case.txt  
Выберите тип:  
1  
2  
0  
Your Answer: 2  
  
0.003857 seconds  
4441285 operation(s)  
12035 input data
```

16047 элементов в считываемом массиве.

```
Укажите путь к файлу: data_16000_worst_case.txt  
Выберите тип:  
1  
2  
0  
Your Answer: 2  
  
0.004867 seconds  
5921689 operation(s)  
16047 input data
```

20059 элементов в считываемом массиве.

```
Укажите путь к файлу: data_20000_worst_case.txt  
Выберите тип:  
1  
2  
0  
Your Answer: 2  
  
0.007301 seconds  
7402093 operation(s)  
20059 input data
```

40117 элементов в считываемом массиве.

Укажите путь к файлу: data_40000_worst_case.txt

Выберите тип:

1

2

0

Your Answer: 2

0.008420 seconds

14803379 operation(s)

40117 input data

5.2.Полином

Рассчитаем сложность на основе зависимости количества операций от объёма входных данных. В ходе выполнения работы я подсчитал количество операций и получил такой многочлен для худшего случая:

$$1 + 1 * (\text{maxDigits} + 1) + 2 * \text{maxDigits} + (1 + 1 * (n + 1) + 2 * n + (24 * 1 + (n - 1) * 33 + 1 * n + 4 * n)) * \text{maxDigits} + 1 * \text{maxDigits} + 1 * \text{maxDigits} + (1 + 1 * (\text{BASE} + 1) + 2 * \text{BASE} + (34n - 4 + 5(\text{BASE} - 1)) * \text{BASE}) * \text{maxDigits} + 2 * \text{maxDigits} + 1 + 1 * (\text{BASE} + 1) + 2 * \text{BASE} + \text{BASE} * (6 * \text{BASE}) + 1 + 1 + \log_{\text{BASE}}(\text{max}) * 4 + 1 + 1 + 2 + 1 + 1 * n + 2 * (n - 1) + ((n - 1) * (2 + 1 + 1)) + 1 + 1$$

5.3.Упростим полином

$$2 + 3 * \text{maxDigits} + (-7 + 40n) * \text{maxDigits} + 2 * \text{maxDigits} + (32 + 340n) * \text{maxDigits} + 2 * \text{maxDigits} + 632 + 2 + 4 * \log_{\text{BASE}}(\text{max}) + 8n = 2 + (370n + 30) * \text{maxDigits} + 8n + 632 + 4 * \log_{\text{BASE}}(\text{max})$$

Константы дают сложность $O(1)$.

$\text{BASE} = 10$ (константа).

$\text{maxDigits} = \log_{10}(\text{max})$, где значение max ограничено диапазоном значений типа `int`.

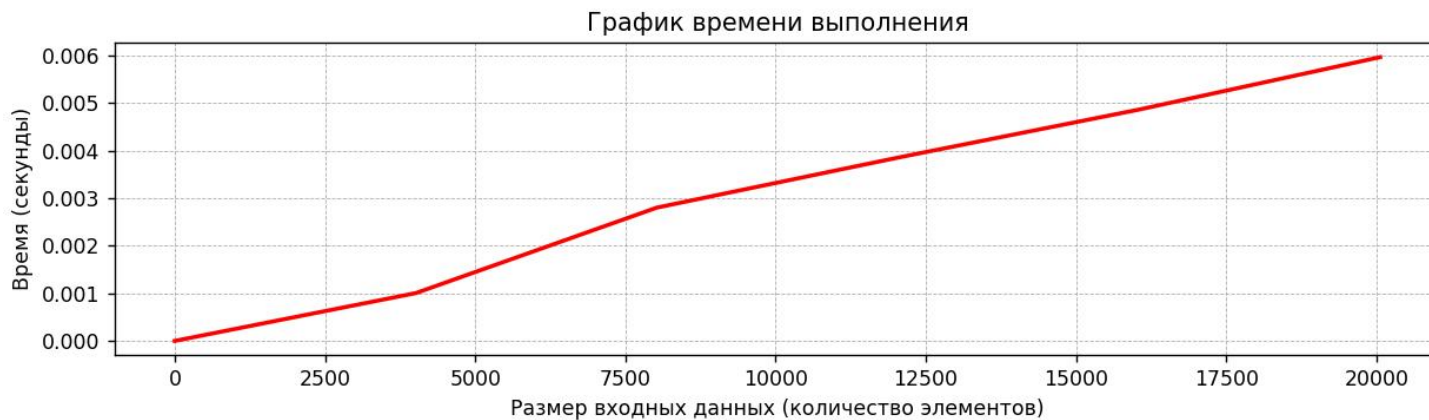
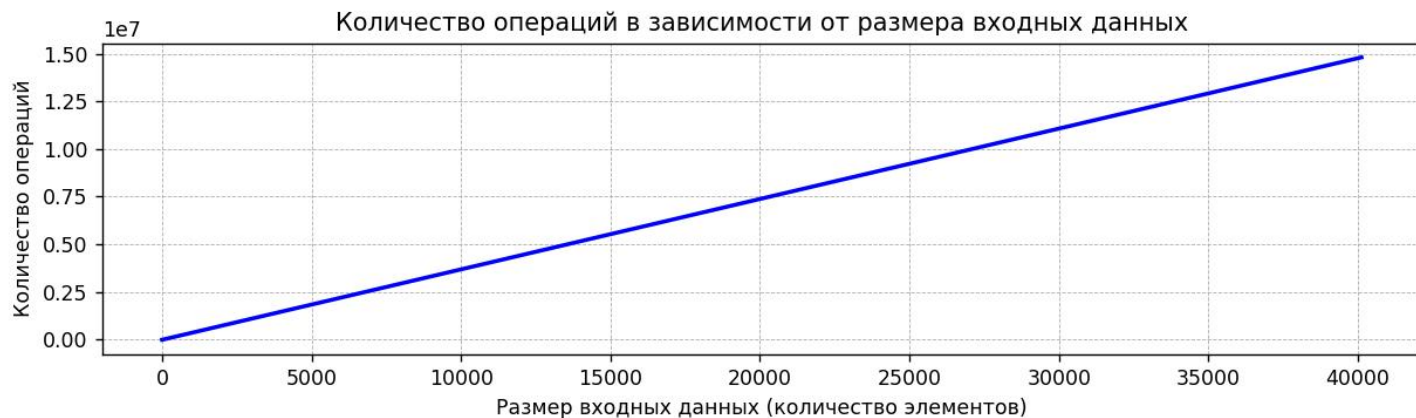
$\log_{10}(\text{max})$ — количество разрядов в максимальном числе.

Таким образом, сложность сводится к $O(n * k)$, где:

- n - количество элементов;
- k - количество разрядов в максимальном числе;
- $\log_{10}(\text{max})$ не зависит от объёма входных данных, т.е. от n .

5.4.Итог

Алгоритм LSD-сортировки на основе циклических очередей имеет сложность, близкую к линейной, что подтверждается также экспериментальными данными.



6. ТЕСТИРОВАНИЕ

Неверный формат опции выбора массива.

```
vboxuser@l111:~/Downloads$ ./Lab1
Код принимает данные из текстового файла в десятичном формате.
Числа могут разделяться пробелами или запятыми.
Укажите путь к файлу: inc.txt

Выберите тип массива:
1 - Статический
2 - Динамический
0 - Выход
Ваш выбор: fg
Некорректный ввод!Попробуйте снова.

Выберите тип массива:
1 - Статический
2 - Динамический
0 - Выход
Ваш выбор: 
```

inc.txt x

12, -7, 5678899, -3, 9
-10, 20, 30, -1

Обработка переполнения статического массива.

```
0 - Выход
Ваш выбор: ^Z
[1]+  Stopped                  ./Lab1
vboxuser@l111:~/Downloads$ ./Lab1
Код принимает данные из текстового файла в десятичном формате.
Числа могут разделяться пробелами или запятыми.
Укажите путь к файлу: t1.txt

Выберите тип массива:
1 - Статический
2 - Динамический
0 - Выход
Ваш выбор: 1
Ошибка: Превышен лимит статического массива!

Выберите тип массива:
1 - Статический
2 - Динамический
0 - Выход
Ваш выбор: 
```

t1.txt x

3423 324 435 4546 547
345 56 4575 67 567 76
3423 324 435 4546 547
345 56 4575 67 567 76
3423 324 435 4546 547
345 56 4575 67 567 76
3423 324 435 4546 547
345 56 4575 67 567 76
3423 324 435 4546 547
345 56 4575 67 567 76
3423 324 435 4546 547
345 56 4575 67 567 76
3423 324 435 4546 547

Сортировка чисел, разделанных запятыми и пробелами.

```
Укажите путь к файлу: t1.txt

Выберите тип массива:
1 - Статический
2 - Динамический
0 - Выход
Ваш выбор: 2

Результат сортировки:
-5 2 3 4 6 7 9
```

Open v

t1.txt
~/Downloads

2 4, 3,9
7, 6, -5

Обработка файла с неверным форматом данных.

Результат сортировки:

-4 -3 6 9 21 34 54

vboxuser@l1111:~/Downloads\$./Lab1

Код принимает данные из текстового файла в десятичном виде.
Числа могут разделяться пробелами или запятыми.

Укажите путь к файлу: inc.txt

Выберите тип массива:

1 - Статический

2 - Динамический

0 - Выход

Ваш выбор: 1

Ошибка: обнаружены некорректные данные в файле!

Укажите путь к файлу:

Open 

inc.txt

×

21 54 -3, 9

34, 6, -4 ss

Сортировка положительных и отрицательных чисел из файла.

Укажите путь к файлу: inc.txt

Выберите тип массива:

1 - Статический

2 - Динамический

0 - Выход

Ваш выбор: 1

Результат сортировки:

-5 -5 -3 2 6 6 23 34 45

inc.txt

×

23 45 34 -5

6 -3 2 -5 6

Обработка чисел в экспоненциальном виде.

```
vboxuser@l111:~/Downloads$ ./Lab1
```

Код принимает данные из текстового файла в десятичном формате.

Числа могут разделяться пробелами или запятыми.

Укажите путь к файлу: ex[.txt

Выберите тип массива:

1 - Статический

2 - Динамический

0 - Выход

Ваш выбор: 1

Результат сортировки:

-45 34 34 200

ex[.txt x

-45 34

34

2e2

7.ВЫВОДЫ

Разработанная программа цифровой сортировки методом LSD реализована на языке C с применением кольцевых очередей на базе односвязного списка. Такая структура обеспечивает динамическое размещение данных. Алгоритм протестирован на различных сценариях и показал корректность и устойчивость работы. Оценка алгоритма осуществлена, блок-схемы подтверждают обоснованность реализации. Все задачи выполнены: чтение и нормализация входных данных, поразрядная сортировка, коррекция отрицательных чисел, тестирование и вывод результата.

