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 №1

«Программа цифровой сортировки с помощью алгоритма LSD»

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.

Это позволяет сортировать обе группы отдельно, а затем объединять результат.

- **Определение числа разрядов.**В каждой группе находится наибольшее по модулю значение M.Число проходов по разрядам d определяется количеством цифр в десятичном представлении M (например, 802 → 3 прохода).

- **Основной цикл поразрядной сортировки.**Повторять d раз для каждой группы отдельно (сначала — отрицательных, потом — неотрицательных):

  - **Распределение.**Для каждого элемента x вычисляется его текущая цифра: digit = (x / 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 → 3 разряда → exp = 1, 10, 100.

3. Иллюстрация работы корзины для digit = 0 (единицы) :

Инициализация:

Создаем bucket0: tail = NULL, size = 0.

Первый элемент (170):

digit = 0.

Создается первый узел с data = 170.

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

Второй элемент (90):

digit = 0.

Создается новый узел (new) с data = 90.

Поскольку tail != NULL, вставляем сразу после tail:

new->next = tail->next (голова очереди с 170),

tail->next = new,

tail = new,

size увеличивается до 2.

Состояние после вставок:

bucket0.size = 2,

bucket0.tail->data = 90,

bucket0.tail->next->data = 170,

(образуется замкнутое кольцо: 170 → 90 → 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].

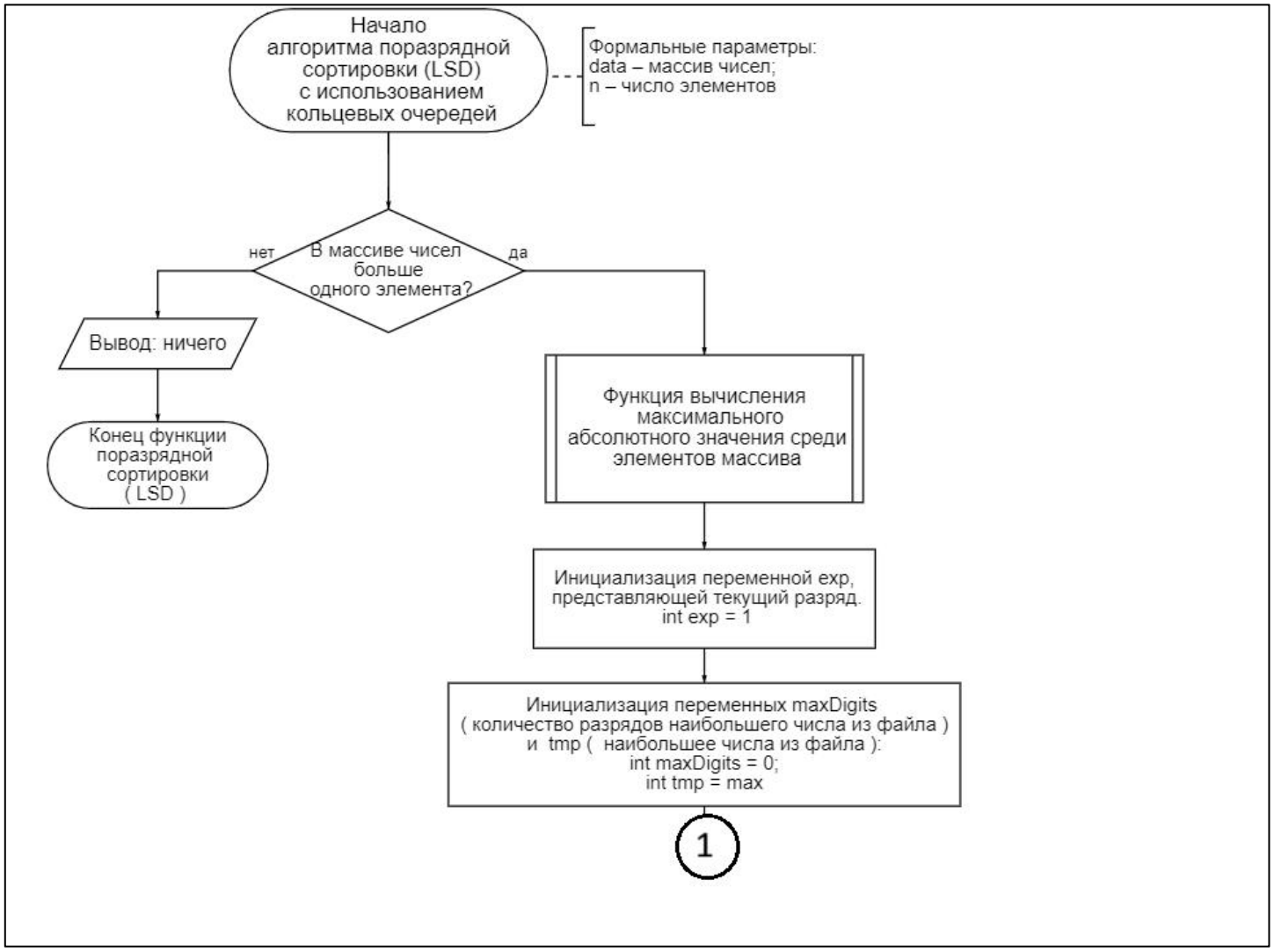
6.Формирование итогового массива

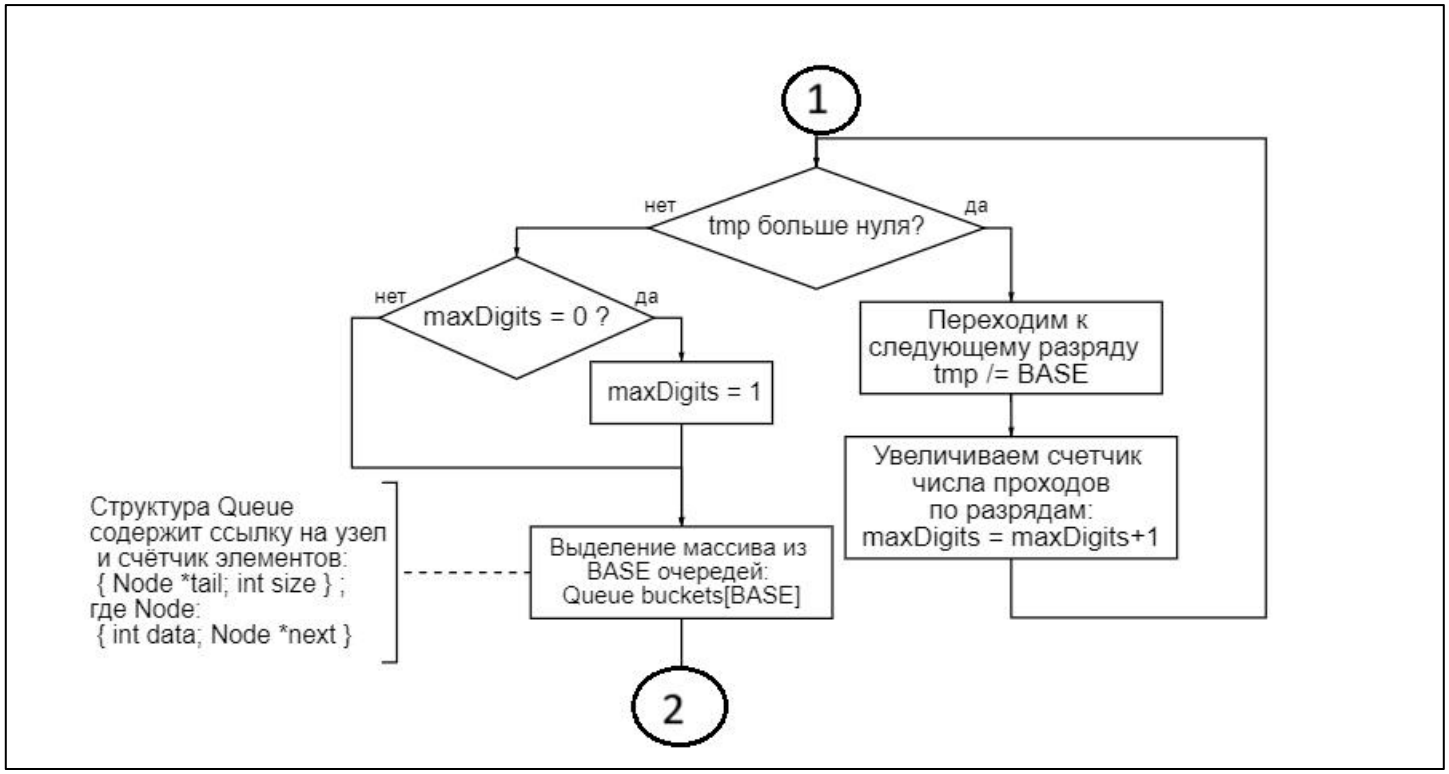
Объединяем отрицательные и положительные: [-802, -75, -66, 2, 24, 45, 90, 170].

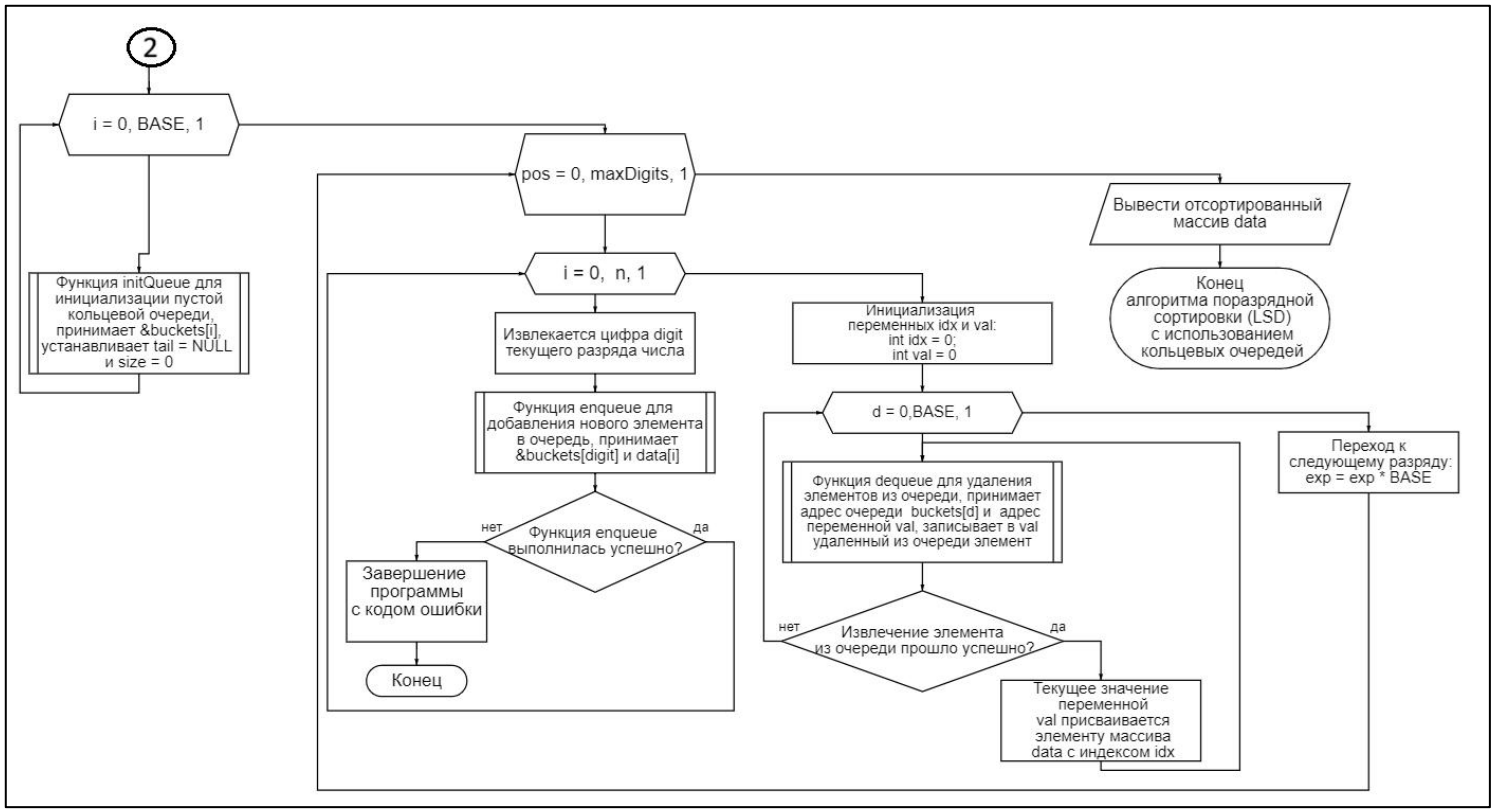
2. ТАБЛИЦА ИСПОЛЬЗУЕМЫХ ПЕРЕМЕННЫХ

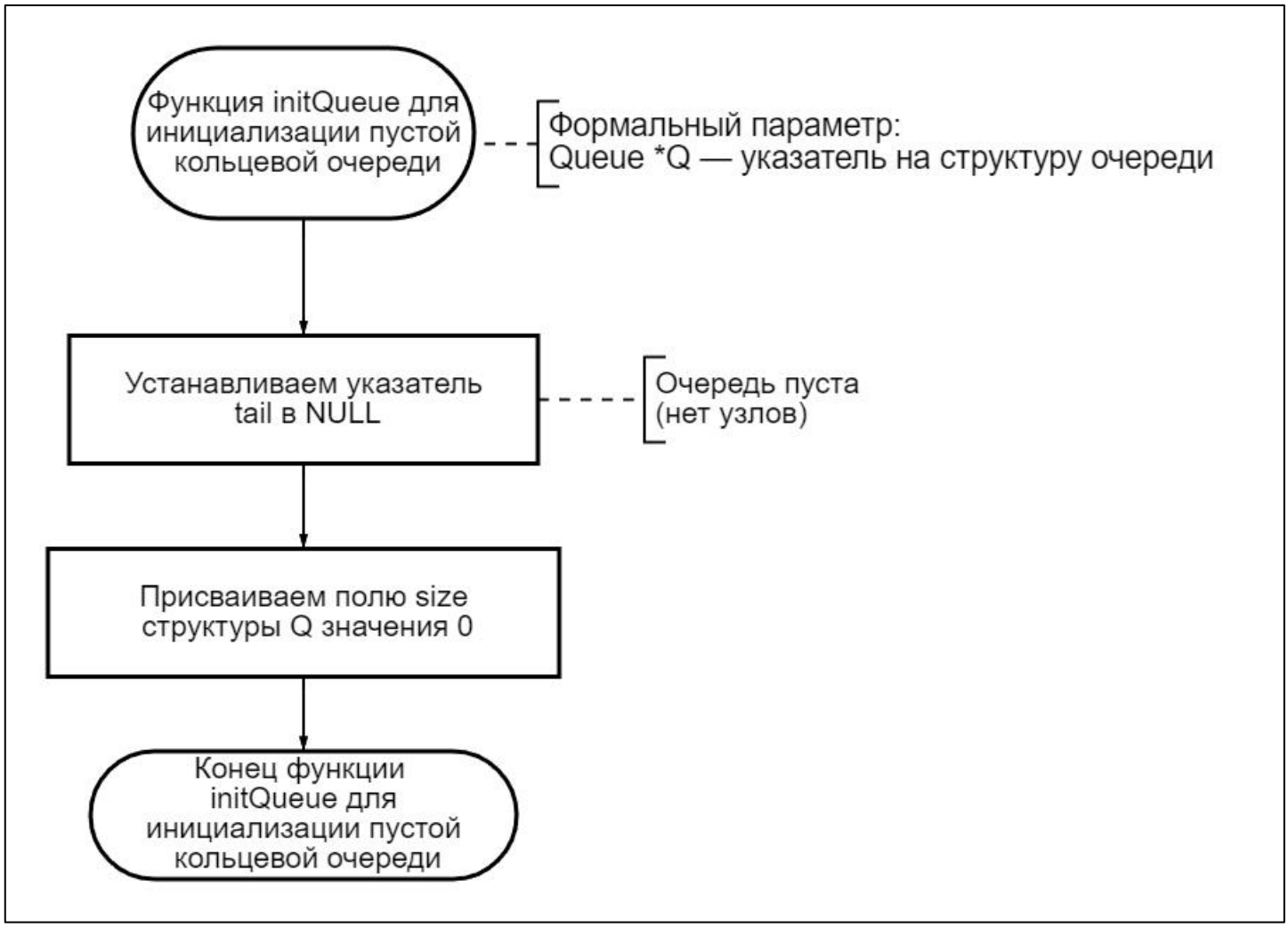
Имя	Тип	Диапазон	Роль в программе
input_file	FILE *	$[0; 2^{64} - 1]$	Указатель на входной файл
filename	char[256]	[0; 255]	Буфер для имени файла
buffer	char[MAX_LINE_LENGTH]	$[-2^7; 2^7 - 1]$	Буфер для чтения строки
storage_type	int	{0, 1, 2}	Выбор типа массива
item_count	int	$[0; 2^{31} - 1]$	Количество прочитанных чисел
capacity	int	$[0; 2^{31} - 1]$	Начальная ёмкость динамического массива
static_data	int[STATIC_SIZE]	$[0; 2^{31} - 1]$	Статический массив
dynamic_data	int *	$[0; 2^{64} - 1]$	Указатель на динамический массив
token	char *	$[0; 2^{64} - 1]$	Токен при разбиении строки
num	int	$[-2^{31}; 2^{31} - 1]$	Преобразованное число
negative_values	int *	$[0; 2^{64} - 1]$	Массив для отрицательных значений (abs)
positive_values	int *	$[0; 2^{64} - 1]$	Массив для неотрицательных значений
neg_cnt	int	$[0; 2^{31} - 1]$	Число отрицательных элементов
pos_cnt	int	$[0; 2^{31} - 1]$	Число неотрицательных элементов
max	int	$[0; 2^{31} - 1]$	Максимальное абсолютное значение
exp	int	$[0; 2^{31} - 1]$	Текущий разряд (1, 10, 100…)
idx	int	$[0; 2^{31} - 1]$	Индекс при сборке
value	long / double	$[-2^{63}; 2^{63} - 1]$ / $[-1.7 \cdot 10^{308}; 1.7 \cdot 10^{308}]$	Переменная при strtol / при strtod
buckets	Queue[BASE]	sizeof(Queue)·BASE	Массив кольцевых очередей Для поразрядной сортировки
maxDigits	int	[1; 10]	Число проходов по разрядам
tmp	int	[0; max]	Вспомогательная переменная при вычислении разрядов
pos	int	[0; maxDigits-1]	Индекс текущего разряда в цикле сортировки
val	int	$[0; 2^{31} - 1]$	Переменная для хранения извлеченного из очереди значения
Q	Queue *Q	$[0; 2^{64} - 1]$	Указатель, параметр функций initQueue, enqueue, dequeue

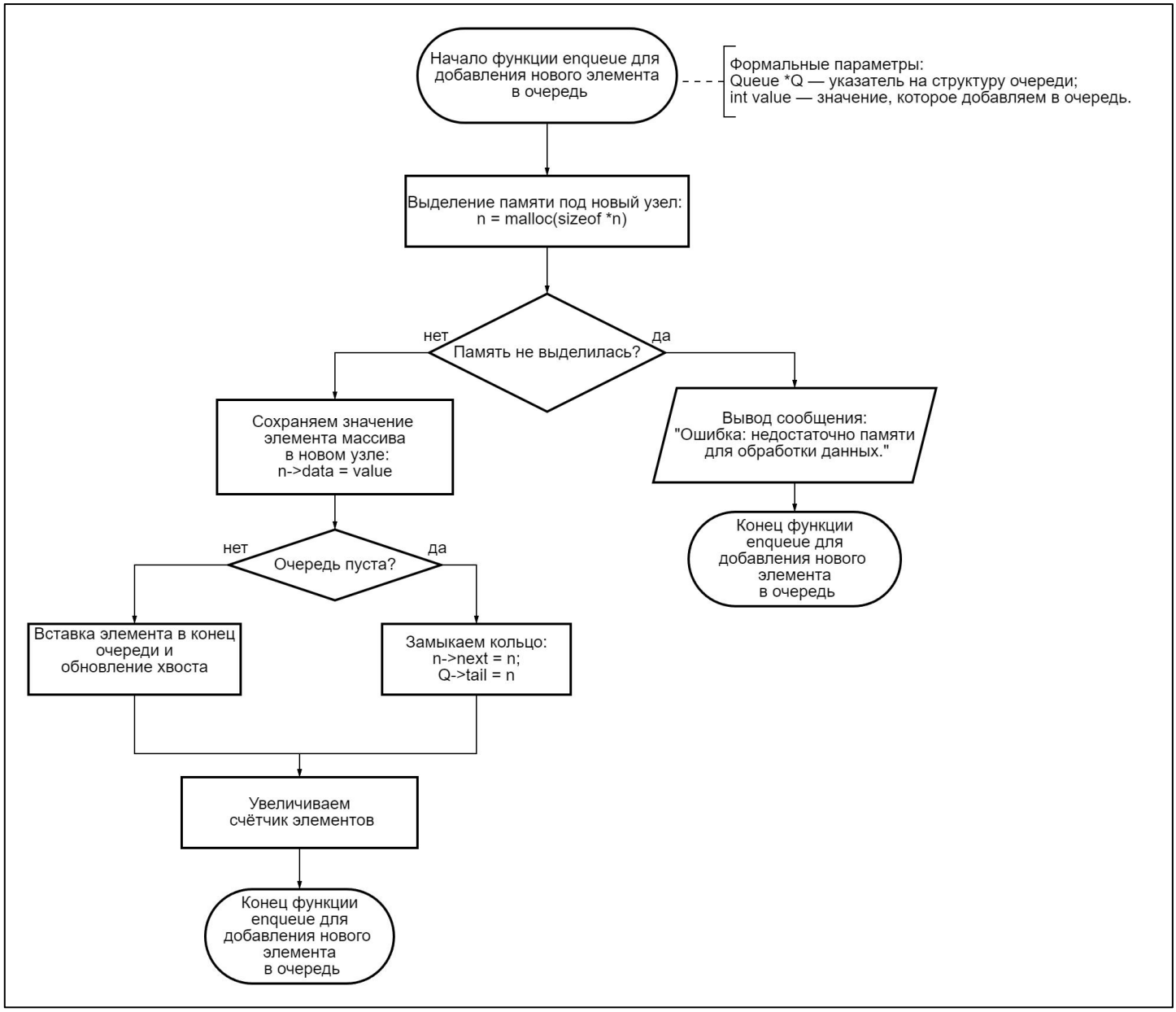
3.БЛОК-СХЕМЫ

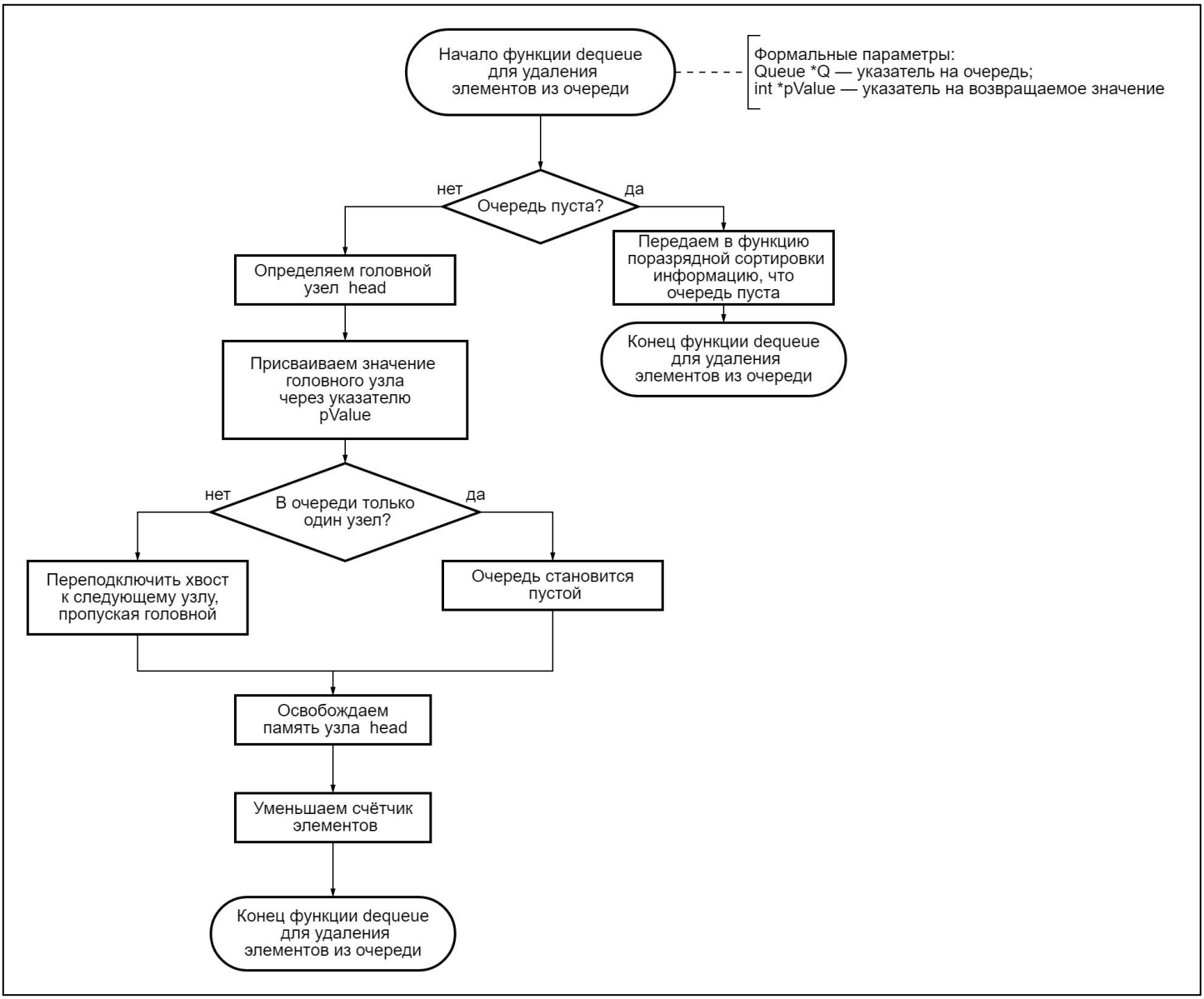

Начало алгоритма поразрядной сортировки (LSD) с использованием кольцевых очередей
Формальные параметры:
data – массив чисел;
n – число элементов
В массиве чисел больше одного элемента?
нет
да
Вывод: ничего
Конец функции поразрядной сортировки (LSD)
Функция вычисления максимального абсолютного значения среди элементов массива
Инициализация переменной exp, представляющей текущий разряд.
int exp = 1
Инициализация переменных maxDigits
(количество разрядов наибольшего числа из файла)
и tmp (наибольшее числа из файла):
int maxDigits = 0;
int tmp = max
1

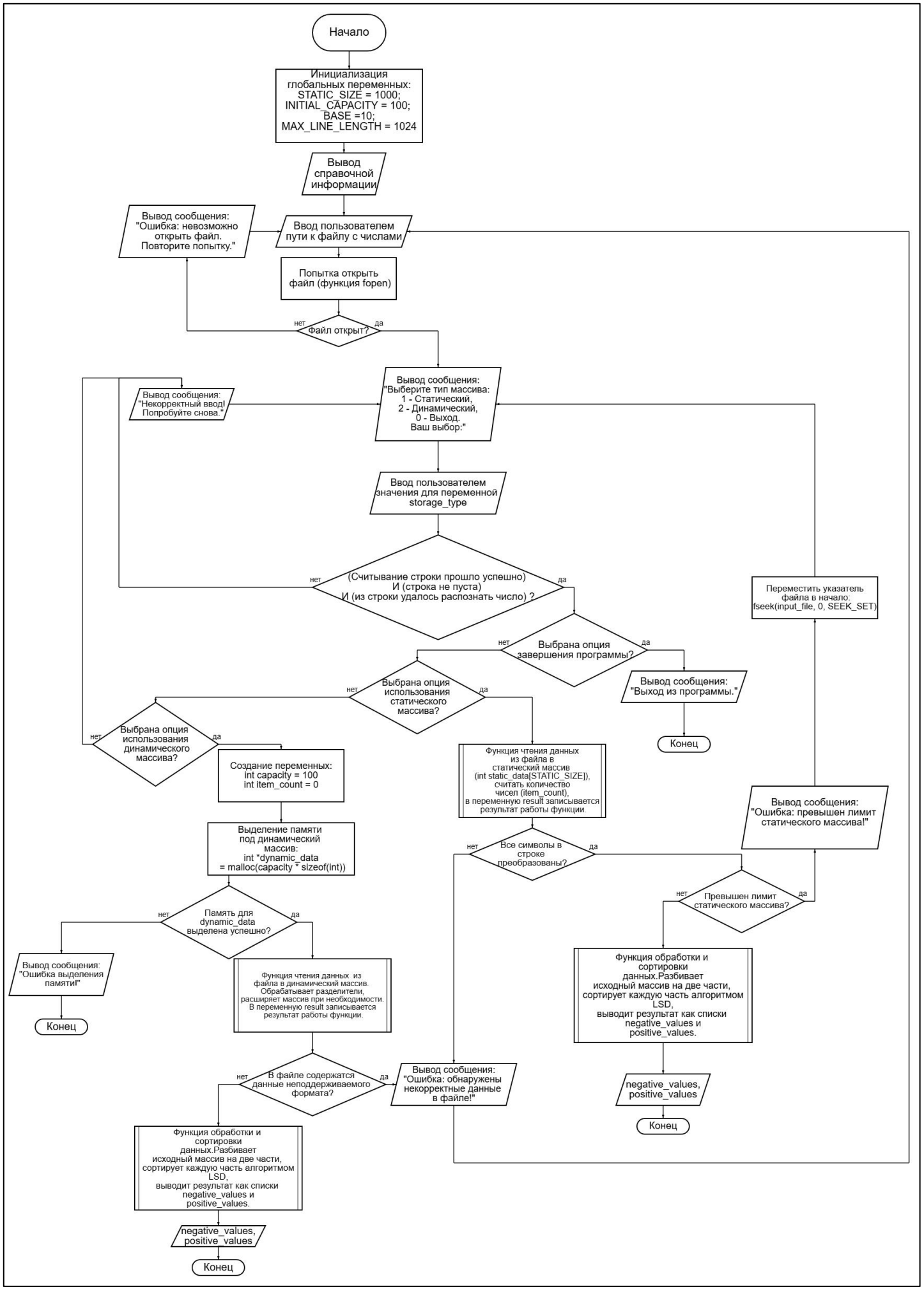

1
tmp больше нуля?
нет
да
maxDigits = 0 ?
нет
да
maxDigits = 1
Переходим к следующему разряду
tmp /= BASE
Увеличиваем счетчик числа проходов по разрядам:
maxDigits = maxDigits+1
Структура Queue
содержит ссылку на узел
и счётчик элементов:
{ Node *tail; int size } ;
где Node:
{ int data; Node *next }
Выделение массива из BASE очередей:
Queue buckets[BASE]
2

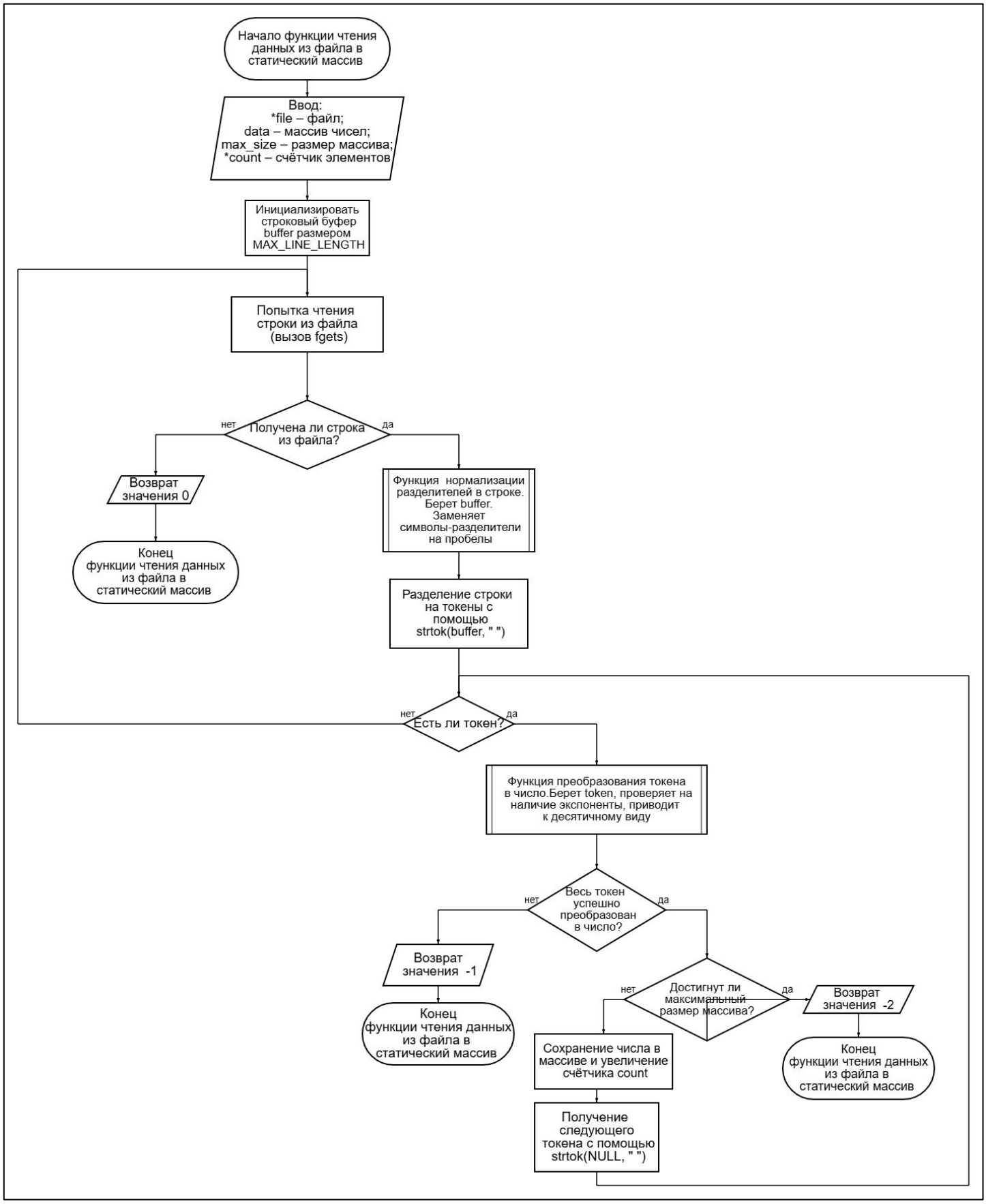

2
i = 0, BASE, 1
Функция initQueue для инициализации пустой кольцевой очереди, принимает &buckets[i], устанавливает tail = NULL и size = 0
pos = 0, maxDigits, 1
Вывести отсортированный массив data
Конец алгоритма поразрядной сортировки (LSD) с использованием кольцевых очередей
i = 0, n, 1
Извлекается цифра digit текущего разряда числа
Функция enqueue для добавления нового элемента в очередь, принимает &buckets[digit] и data[i]
нет
Функция enqueue выполнилась успешно?
да
Завершение программы с кодом ошибки
Конец
Инициализация переменных idx и val:
int idx = 0;
int val = 0
d = 0,BASE, 1
Переход к следующему разряду:
exp = exp * BASE
Функция dequeue для удаления элементов из очереди, принимает адрес очереди buckets[d] и адрес переменной val, записывает в val удаленный из очереди элемент
нет
Извлечение элемента из очереди прошло успешно?
да
Текущее значение переменной val присваивается элементу массива data с индексом idx

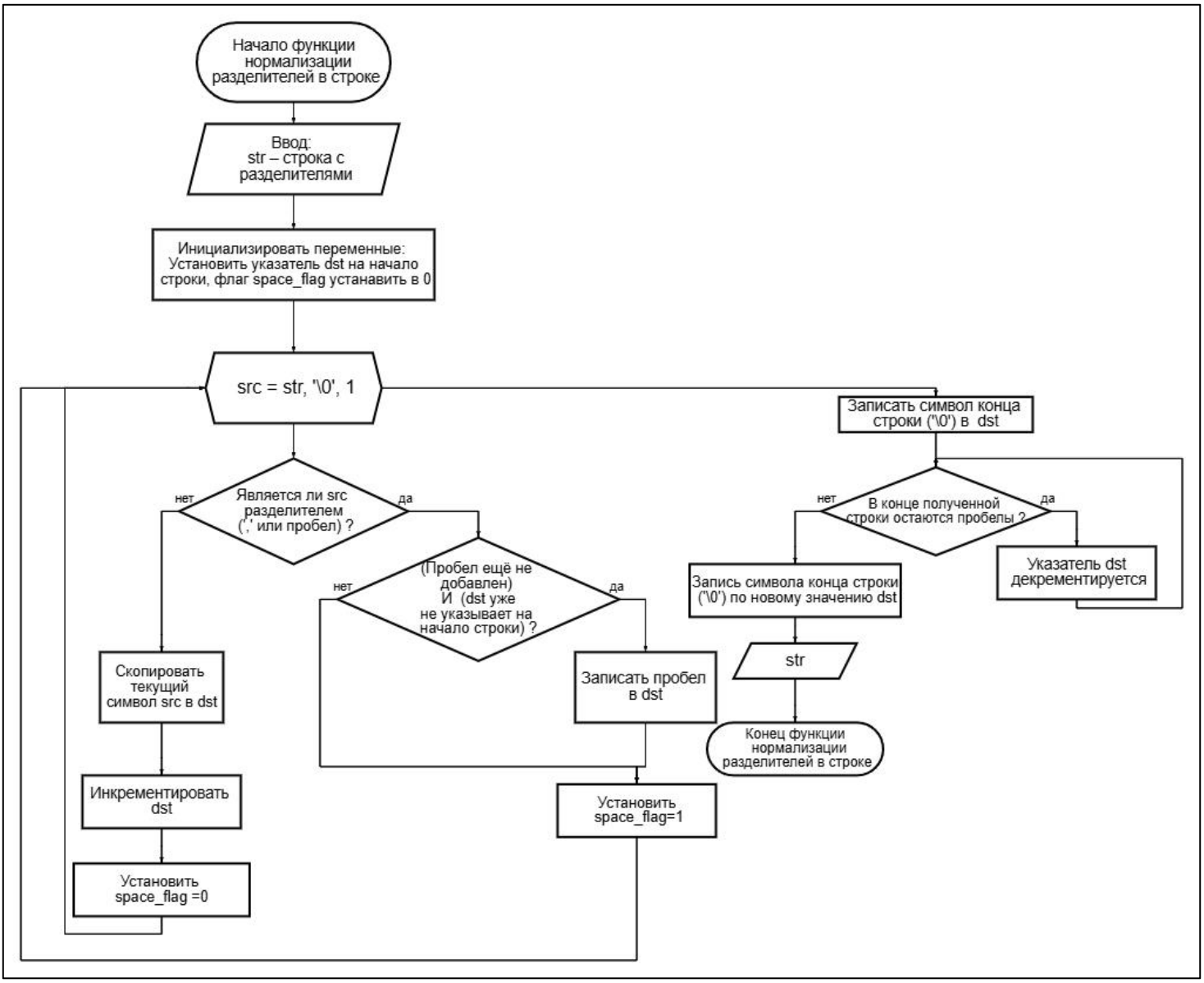

Функция initQueue для инициализации пустой кольцевой очереди
Формальный параметр:
Queue *Q — указатель на структуру очереди
Устанавливаем указатель tail в NULL
Очередь пуста (нет узлов)
Присваиваем полю size структуры Q значения 0
Конец функции initQueue для инициализации пустой кольцевой очереди

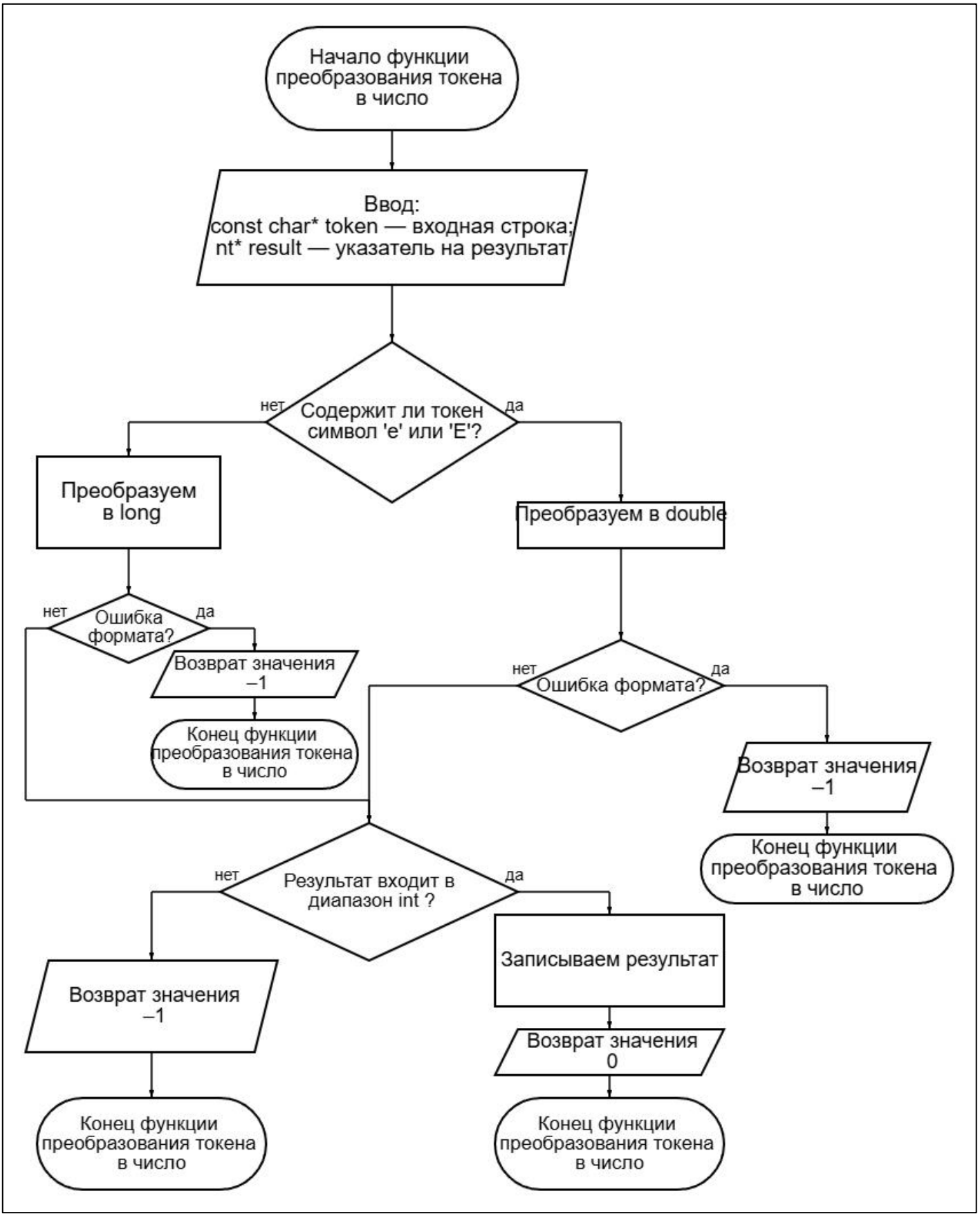

Начало функции enqueue для добавления нового элемента в очередь
Формальные параметры:
Queue *Q — указатель на структуру очереди;
int value — значение, которое добавляем в очередь.
Выделение памяти под новый узел:
n = malloc(sizeof *n)
нет
Память не выделилась?
да
Сохраняем значение элемента массива в новом узле:
n->data = value
Вывод сообщения:
"Ошибка: недостаточно памяти для обработки данных."
Конец функции enqueue для добавления нового элемента в очередь
нет
Очередь пуста?
да
Вставка элемента в конец очереди и обновление хвоста
Замыкаем кольцо:
n->next = n;
Q->tail = n
Увеличиваем счётчик элементов
Конец функции enqueue для добавления нового элемента в очередь

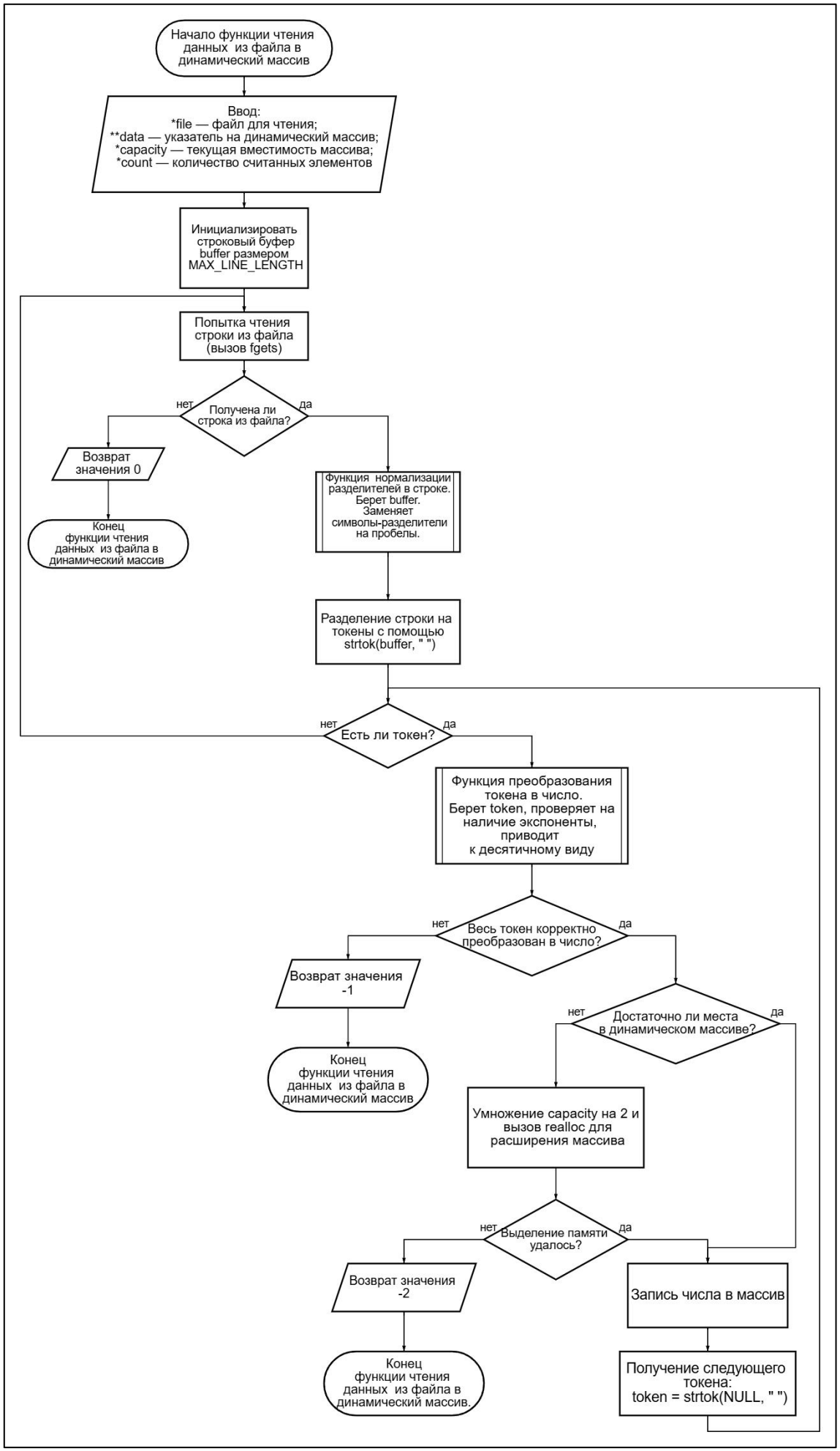

Начало функции dequeue для удаления элементов из очереди
Формальные параметры:
Queue *Q — указатель на очередь;
int *pValue — указатель на возвращаемое значение
Очередь пуста?
нет
да
Определяем головной узел head
Передаем в функцию поразрядной сортировки информацию, что очередь пуста
Конец функции dequeue для удаления элементов из очереди
Присваиваем значение головного узла через указателю pValue
В очереди только один узел?
нет
да
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

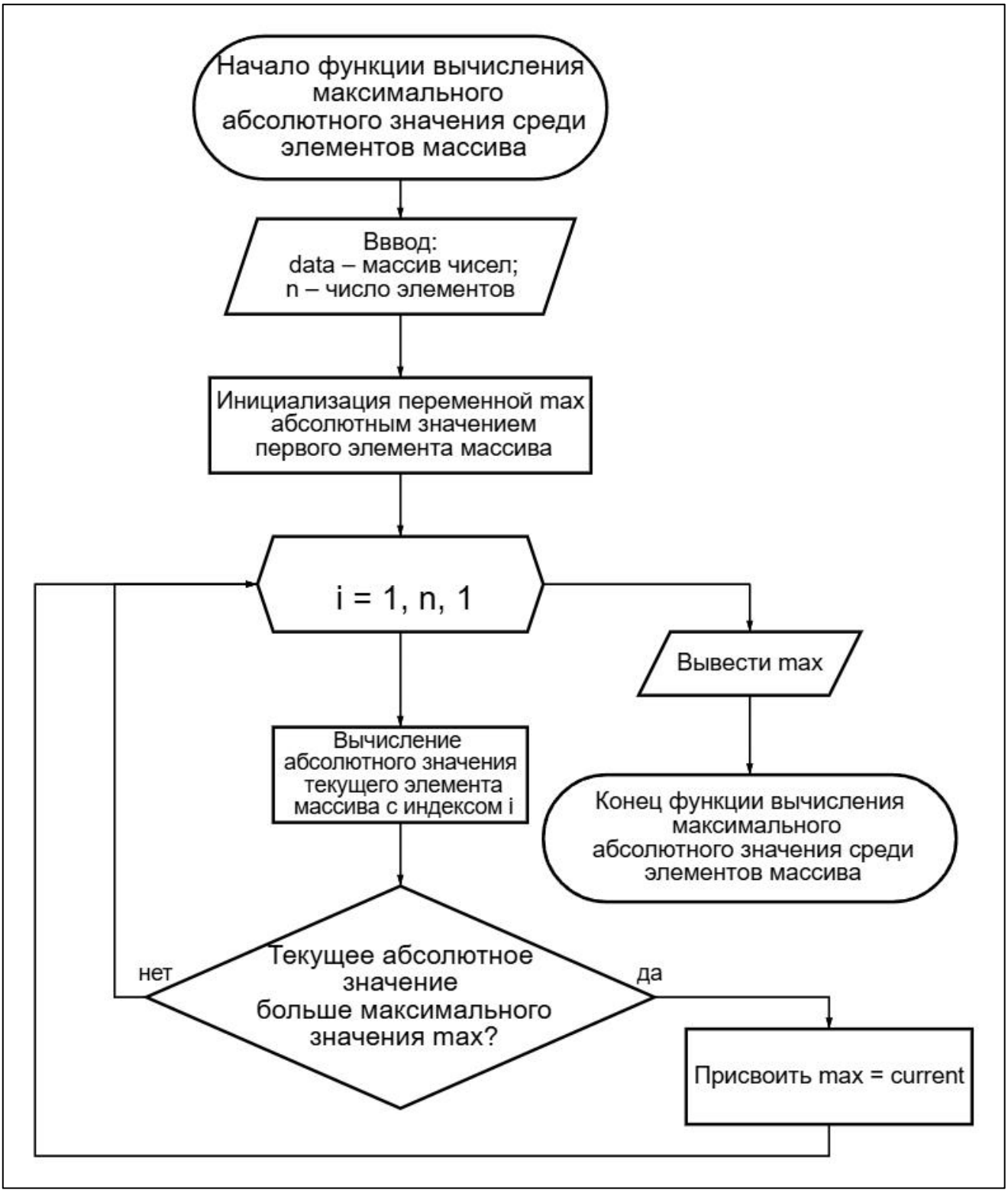

Начало
Инициализация глобальных переменных: STATIC_SIZE = 1000; INITIAL_CAPACITY = 100; BASE =10; MAX_LINE_LENGTH = 1024
Вывод справочной информации
Вывод сообщения: "Ошибка: невозможно открыть файл. Повторите попытку."
Ввод пользователем пути к файлу с числами
Попытка открыть файл (функция fopen)
нет
Файл открыт?
да
Вывод сообщения: "Некорректный ввод! Попробуйте снова."
Вывод сообщения: "Выберите тип массива: 1 - Статический, 2 - Динамический, 0 - Выход. Ваш выбор:"
Ввод пользователем значения для переменной storage_type
нет
(Считывание строки прошло успешно) И (строка не пуста) И (из строки удалось распознать число) ?
да
нет
Выбрана опция завершения программы?
да
Вывод сообщения: "Выход из программы."
Конец
нет
Выбрана опция использования статического массива?
да
Функция чтения данных из файла в статический массив (int static_data[STATIC_SIZE]), считать количество чисел (item_count), в переменную result записывается результат работы функции.
нет
Все символы в строке преобразованы?
да
Переместить указатель файла в начало: fseek(input_file, 0, SEEK_SET)
Вывод сообщения: "Ошибка: превышен лимит статического массива!"
нет
Превышен лимит статического массива?
да
Функция обработки и сортировки данных.Разбивает исходный массив на две части, сортирует каждую часть алгоритмом LSD, выводит результат как списки negative_values и positive_values.
negative_values, positive_values
Конец
нет
Выбрана опция использования динамического массива?
да
Создание переменных: int capacity = 100 int item_count = 0
Выделение памяти под динамический массив: int *dynamic_data = malloc(capacity * sizeof(int))
нет
Память для dynamic_data выделена успешно?
да
Вывод сообщения: "Ошибка выделения памяти!"
Конец
Функция чтения данных из файла в динамический массив. Обрабатывает разделители, расширяет массив при необходимости. В переменную result записывается результат работы функции.
нет
В файле содержатся данные неподдерживаемого формата?
да
Вывод сообщения: "Ошибка: обнаружены некорректные данные в файле!"
Функция обработки и сортировки данных.Разбивает исходный массив на две части, сортирует каждую часть алгоритмом LSD, выводит результат как списки negative_values и positive_values.
negative_values, positive_values
Конец

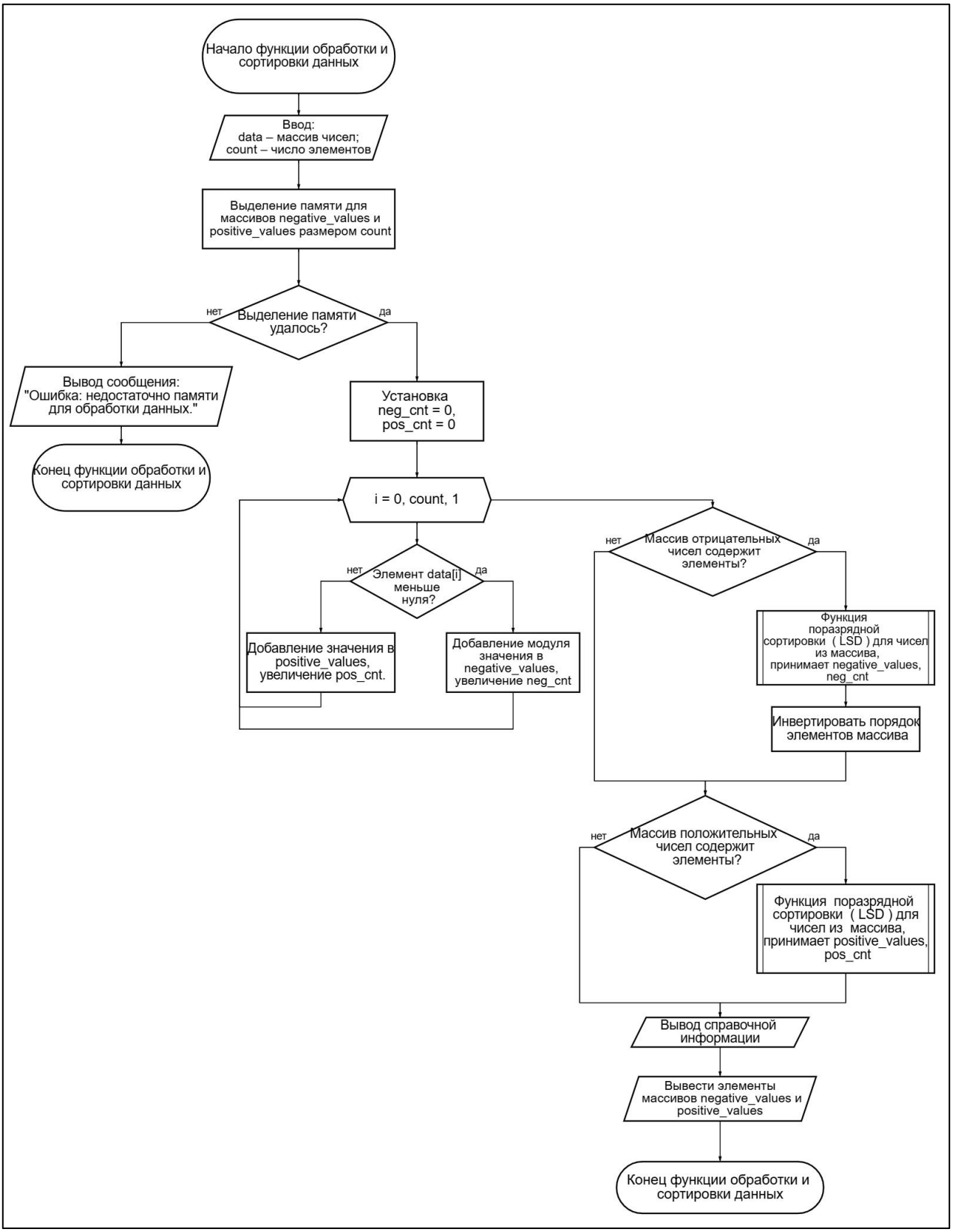

Начало функции чтения данных из файла в статический массив
Ввод:
*file – файл;
data – массив чисел;
max_size – размер массива;
*count – счётчик элементов
Инициализировать строковый буфер buffer размером MAX_LINE_LENGTH
Попытка чтения строки из файла (вызов fgets)
Получена ли строка из файла?
нет
да
Возврат 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, " ")


Начало функции нормализации разделителей в строке
Ввод: str – строка с разделителями
Инициализировать переменные: Установить указатель dst на начало строки, флаг space_flag устанавить в 0
src = str, '\0', 1
Является ли src разделителем (',' или пробел) ?
нет
да
(Пробел ещё не добавлен) И (dst уже не указывает на начало строки) ?
нет
да
Скопировать текущий символ src в dst
Инкрементировать dst
Установить space_flag =0
Записать пробел в dst
Установить space_flag=1
Записать символ конца строки ('\0') в dst
В конце полученной строки остаются пробелы ?
нет
да
Запись символа конца строки ('\0') по новому значению dst
Указатель dst декрементируется
str
Конец функции нормализации разделителей в строке


Начало функции преобразования токена в число
Ввод:
const char* token — входная строка;
nt* result — указатель на результат
Содержит ли токен символ 'e' или 'E'?
нет
да
Преобразуем в long
Преобразуем в double
Ошибка формата?
нет
да
Возврат значения –1
Конец функции преобразования токена в число
Ошибка формата?
нет
да
Возврат значения –1
Конец функции преобразования токена в число
Результат входит в диапазон int ?
нет
да
Возврат значения –1
Конец функции преобразования токена в число
Записываем результат
Возврат значения 0
Конец функции преобразования токена в число


Начало функции чтения данных из файла в динамический массив
Ввод:
*file — файл для чтения;
**data — указатель на динамический массив;
*capacity — текущая вместимость массива;
*count — количество считанных элементов
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 динамический 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 корректно преобразован в число?
нет
да
Возврат значения -1
Конец функции чтения данных из файла в динамический массив
Достаточно ли места в динамическом массиве?
нет
да
Умножение capacity на 2 и вызов realloc для расширения массива
Выделение памяти удалось?
нет
да
Возврат значения -2
Конец функции чтения данных из файла в динамический массив.
Запись числа в массив
Получение следующего токена: token = strtok(NULL, " ")


Начало функции вычисления максимального абсолютного значения среди элементов массива
Вввод:
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
нет
Текущее абсолютное значение больше максимального значения max?
да
Присвоить max = current


Начало функции обработки и сортировки данных
Ввод:
data – массив чисел;
count – число элементов
Выделение памяти для массивов negative_values и positive_values размером count
нет
Выделение памяти удалось?
да
Вывод сообщения:
"Ошибка: недостаточно памяти для обработки данных."
Конец функции обработки и сортировки данных
Установка neg_cnt = 0, pos_cnt = 0
i = 0, count, 1
нет
Элемент data[i] меньше нуля?
да
Добавление значения в positive_values, увеличение pos_cnt.
Добавление модуля значения в negative_values, увеличение neg_cnt
нет
Массив отрицательных чисел содержит элементы?
да
Функция поразрядной сортировки (LSD) для чисел из массива, принимает negative_values, neg_cnt
Инвертировать порядок элементов массива
нет
Массив положительных чисел содержит элементы?
да
Функция поразрядной сортировки (LSD) для чисел из массива, принимает positive_values, pos_cnt
Вывод справочной информации
Вывести элементы массивов negative_values и positive_values
Конец функции обработки и сортировки данных

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 связного 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. АНАЛИЗ СЛОЖНОСТИ

Теория:

LSD-сортировка относится к неадаптивным алгоритмам с линейной сложностью.
Основная идея: распределение элементов по корзинам (очередям) для каждого разряда.
Сложность LSD-сортировки: **O(d · (n + k))**, где:

d — количество разрядов в максимальном числе,
n — количество элементов,
k — количество корзин (в моем случае **k = 10**, так как основание системы счисления BASE = 10).

Анализ кода

1. Основные операции

Распределение по корзинам:

Каждый элемент добавляется в очередь (enqueue) для текущего разряда.
Сложность: **O(n)** на каждый разряд.

Сборка из корзин:

Все элементы извлекаются из очередей (dequeue) и объединяются в массив.
Сложность: **O(n)** на каждый разряд.

2. Реализация очереди

Кольцевая очередь на связном списке:

Операции enqueue и dequeue выполняются за **O(1)**, так как не требуют перебора элементов.
Память: **O(n)** для хранения узлов списка.

3. Дополнительные шаги

Обработка отрицательных чисел:

Разделение на два массива — **O(n)**.

Чтение данных из файла:

Зависит от размера входных данных, но выполняется за **O(n)**.

Экспериментальные данные

Из результатов тестирования:

Количество элементов: **1088**.
Количество разрядов: **6**.
Операций enqueue и dequeue: **6528** (1088 · 6 = 6528).
Время выполнения: **0.00041 сек** для 6 разрядов.

Выводы из теста:

Количество операций растет линейно
Время выполнения пропорционально количеству операций, что подтверждает теоретическую оценку.

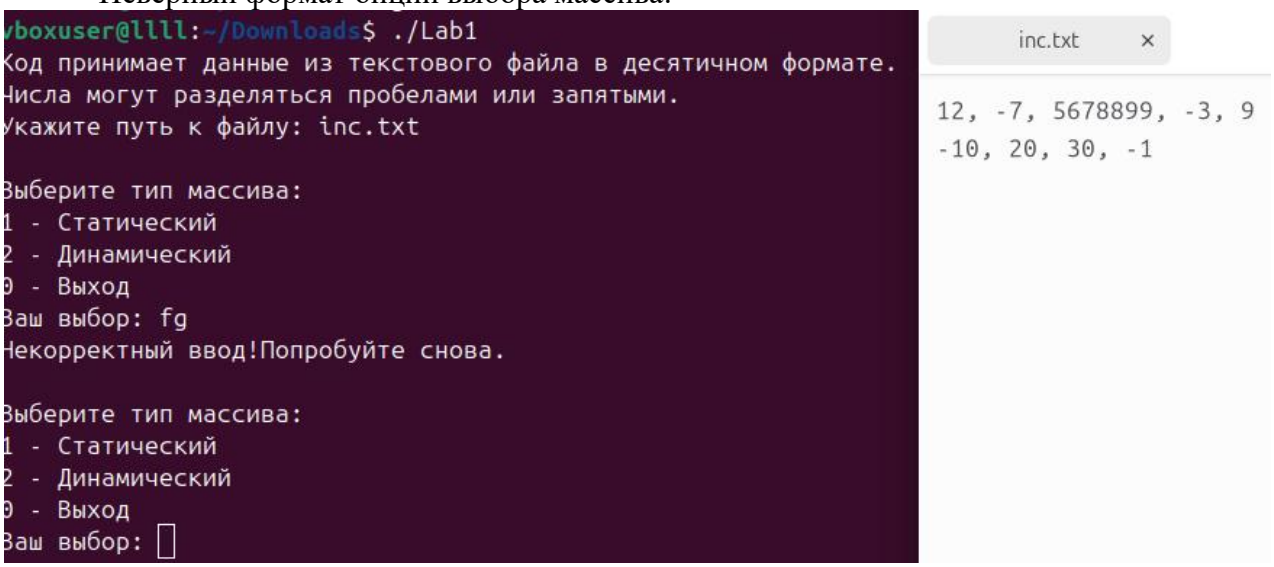
```
Укажите путь к файлу: 1.txt

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

```
 Разряд 0: время 0.00009 сек, enqueue+dequeue = 1088+1088
 Разряд 1: время 0.00016 сек, enqueue+dequeue = 2176+2176
 Разряд 2: время 0.00023 сек, enqueue+dequeue = 3264+3264
 Разряд 3: время 0.00029 сек, enqueue+dequeue = 4352+4352
 Разряд 4: время 0.00035 сек, enqueue+dequeue = 5440+5440
 Разряд 5: время 0.00041 сек, enqueue+dequeue = 6528+6528
```

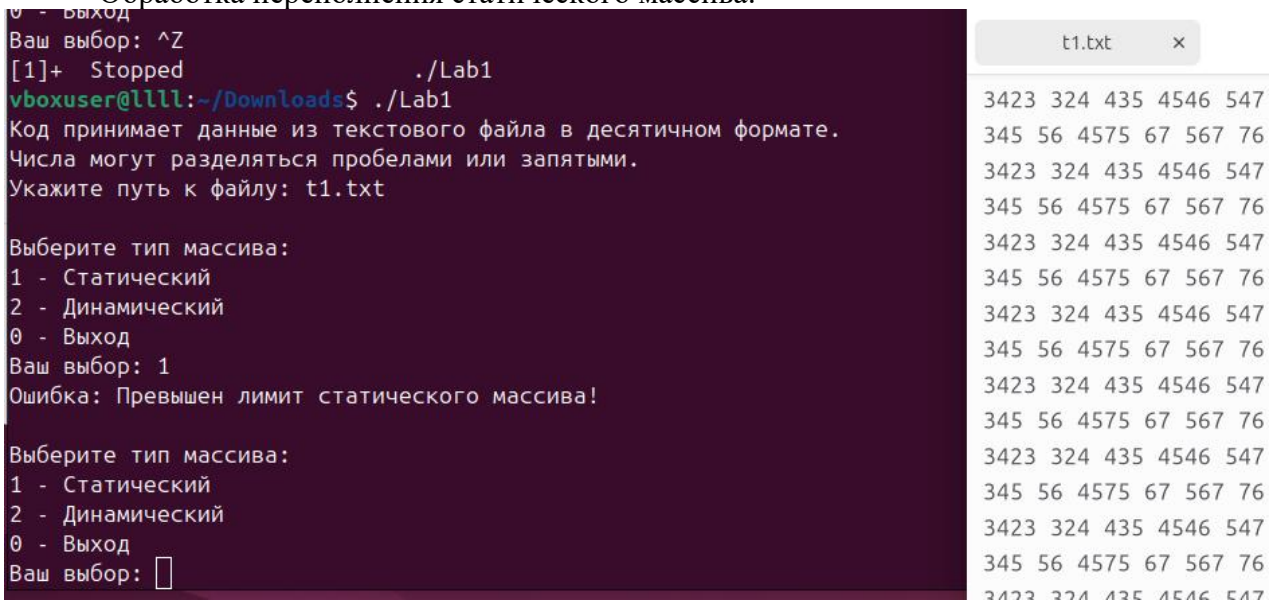
```
Enqueue вызван 6528 раз, Dequeue вызван 6528 раз
```

6.ТЕСТИРОВАНИЕ

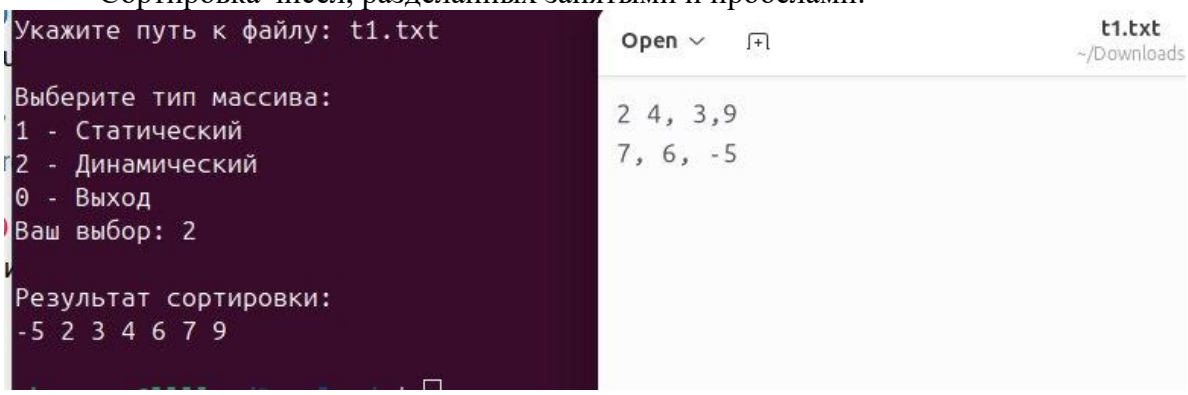
Неверный формат опции выбора массива:



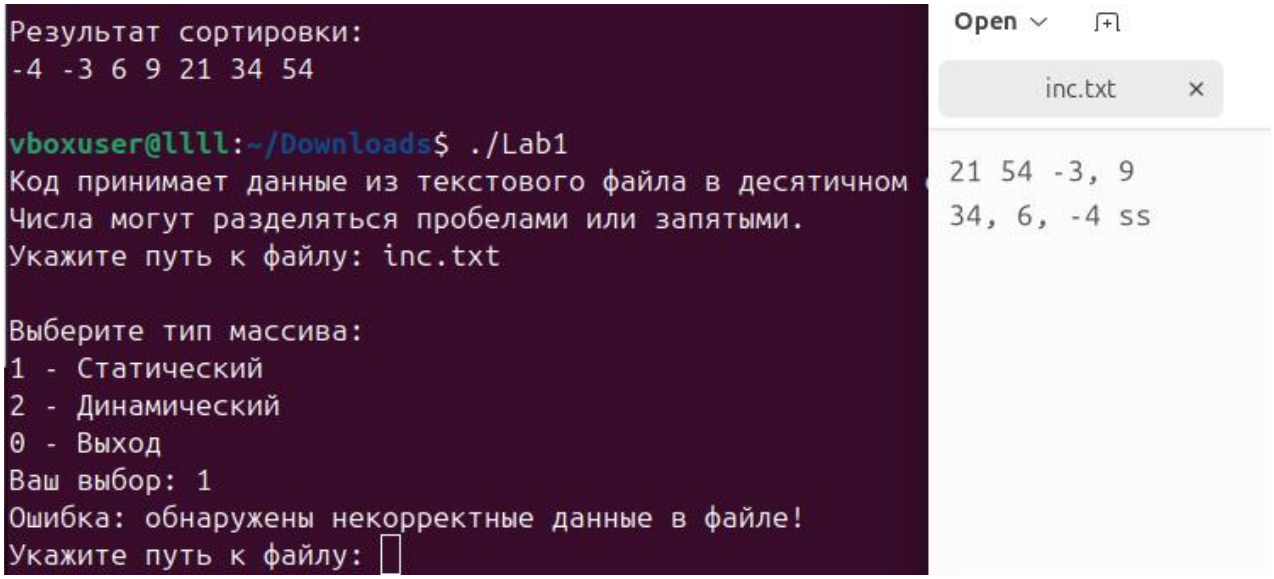
Обработка переполнения статического массива:



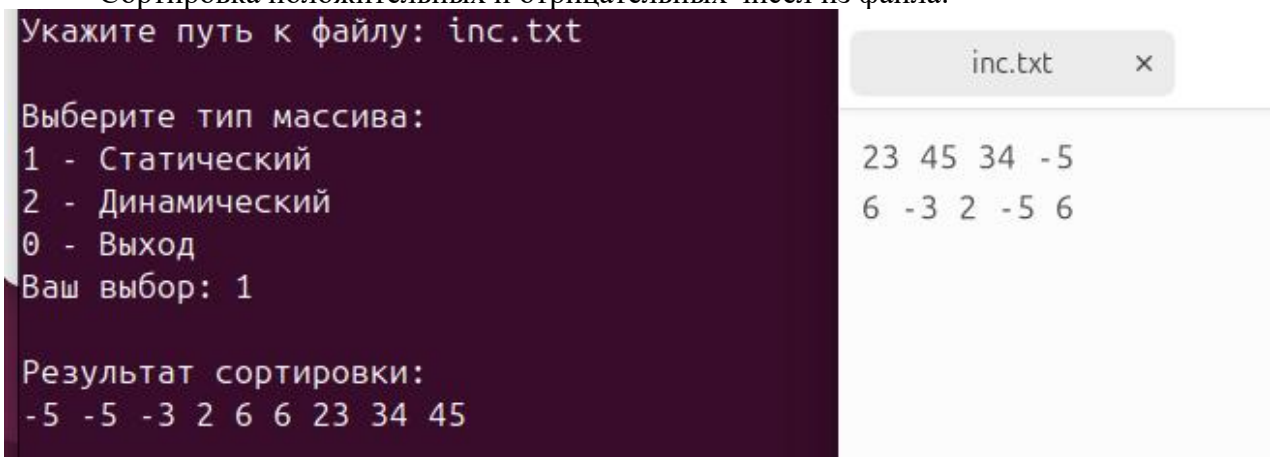
Сортировка чисел, разделанных запятыми и пробелами:



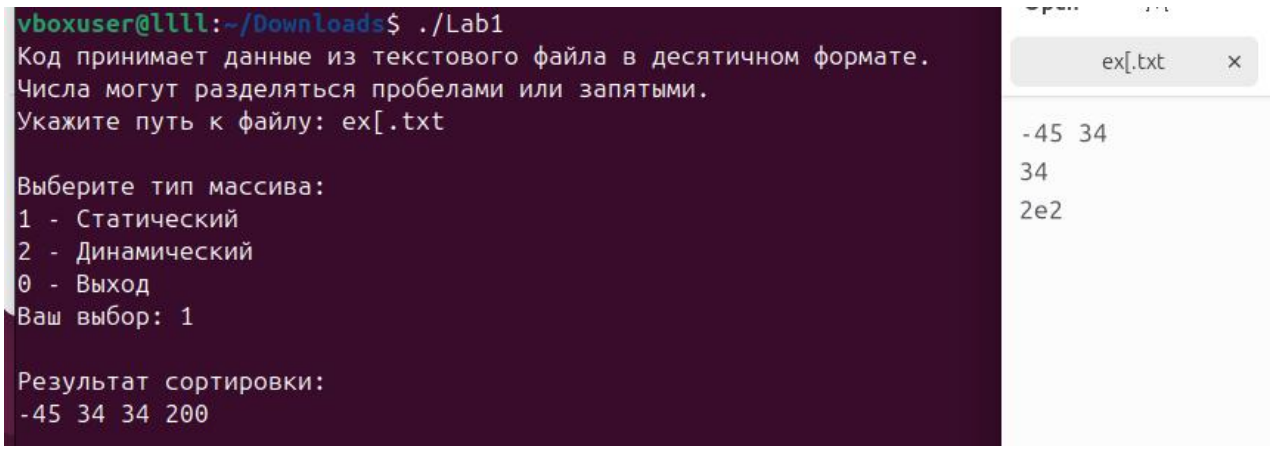
Обработка файла с неверным форматом данных:



Сортировка положительных и отрицательных чисел из файла:



Обработка чисел в экспоненциальном виде:



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 (пустой файл, переполнение статического массива, динамическое расширение, смешанные знаки, экспоненциальная нотация, некорректный ввод) 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.