|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

|  |  |
| --- | --- |
| **Отчет по выполнению практического задания № 5, часть 1** | |
| **Тема:** | |
| **«Работа с данными из файла»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Лисовский И.В. |
|  | Группа: ИКБО-21-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_Toc176566741)

[2 ЗАДАНИЕ №1 4](#_Toc176566742)

[2.1 Формулировка задачи 4](#_Toc176566743)

[2.2 Математическая модель решения (описание алгоритма) для пункта а 4](#_Toc176566744)

[2.3 Код программы с комментариями для пункта а 5](#_Toc176566745)

[2.4 Результаты тестирования для пункта а 5](#_Toc176566746)

[2.5 Математическая модель решения (описание алгоритма) для пункта б 5](#_Toc176566747)

[2.6 Код программы с комментариями для пункта б 6](#_Toc176566748)

[2.7 Результаты тестирования для пункта б 7](#_Toc176566749)

[2.8 Математическая модель решения (описание алгоритма) для пункта в 7](#_Toc176566750)

[2.8 Код программы с комментариями для пункта в 9](#_Toc176566751)

[2.9 Результаты тестирования для пункта в 9](#_Toc176566752)

[2.10 Вывод по заданию №1 9](#_Toc176566753)

[3 ЗАДАНИЕ №2 9](#_Toc176566754)

[3.1 Формулировка задачи 9](#_Toc176566755)

[3.2 Математическая модель решения (описание алгоритма) для пункта а 10](#_Toc176566756)

[3.3 Код программы с комментариями для пункта а 11](#_Toc176566757)

[3.4 Результаты тестирования для пункта а 12](#_Toc176566758)

[3.5 Математическая модель решения (описание алгоритма) для пункта б 12](#_Toc176566759)

[3.6 Код программы с комментариями для пункта б 15](#_Toc176566760)

[3.7 Результаты тестирования для пункта б 15](#_Toc176566761)

[3.8 Математическая модель решения (описание алгоритма) для пункта в 15](#_Toc176566762)

[3.9 Код программы с комментариями для пункта в 17](#_Toc176566763)

[3.10 Результаты тестирования для пункта в 17](#_Toc176566764)

[3.11 Вывод по заданию №2 17](#_Toc176566765)

[4 ВОПРОСЫ ДЛЯ САМОПОДГОТОВКИ 27](#_Toc176566766)

[4 ВЫВОДЫ **Ошибка! Закладка не определена.**](#_Toc176566767)

[5 ЛИТЕРАТУРА 29](#_Toc176566768)

# **1 ЦЕЛЬ**

Изучить методы работы с битовым представлением целых чисел без знака и разработать оптимальный алгоритм внешней сортировки, используя битовый массив.

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи**

1. Реализовать пример из методички, проверить правильность результата в том числе и на других значениях х.
2. Реализовать по аналогии с этим же примером установку 7-го бита числа в единицу.
3. Реализовать код листинга 1 из методички, объясните выводимый программой результат.

# **2.2 Математическая модель решения (описание алгоритма)** **для пункта а**

Исходные данные:

- Используются три 8-разрядных двоичных числа:

* x=255 (в двоичном виде: 11111111)

- Задана 8-разрядная маска:

* maska=1 (в двоичной системе: 00000001)

Цель программы:

Необходимо сбросить (обнулить) 4-й бит в каждом из числа x с использованием побитовых операций и маски.

Описание алгоритма:

1. Инициализация маски:

Маска представляет собой 8-битное число, в котором установлен 1 в младшем разряде: maska=00000001.

2. Сдвиг маски на 4 бита:

Выполняется сдвиг маски влево на 4 позиции, чтобы единичный бит оказался на 5-й позиции:maska≪4=00010000

3. Инверсия маски (~):

Для того чтобы сбросить только нужный бит, необходимо инвертировать полученную маску:

∼(maska≪4)=∼00010000=11101111

В результате инверсии, 5-й бит устанавливается в 0, а все остальные остаются равными 1.

4. Побитовая операция И (&):

Применяем побитовую операцию И между каждым числом и инвертированной маской, чтобы сбросить нужный бит:

x = x& ~(maska<< 4)

* Для x = 11111111 (255 в десятичной системе)

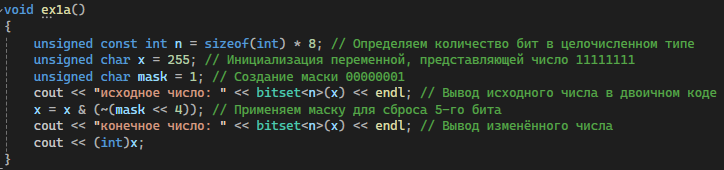
Инвертированная маска: 11101111

Результат: 11111111 & 11101111 = 11101111 (239 в десятичной системе)

5. Вывод результата:

Программа выводит обновлённое значение: x = 239

**2.3Код программы с комментариями для пункта а**



Листинг 1 —Код к пункту а

### **2.4 Результаты тестирования для пункта а**

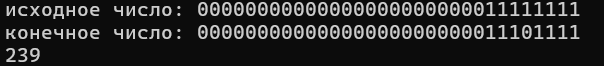


Рисунок 1 — Тестирование кода к пункту

### **2.5 Математическая модель решения (описание алгоритма) для пункта б**

Исходные данные:

- Используется 8-разрядное двоичное число:

* x = 63 (в двоичной системе: 00111111)

- Задана 8-разрядная маска:

* maska = 1 (в двоичной системе: 00000001)

Цель программы:

Необходимо установить значение 1 в шестом бите числа xxx с использованием побитовой операции "ИЛИ".

Описание алгоритма:

1. Инициализация маски:

Маска maska представляет собой 8-разрядное число, в котором установлен 1 в младшем бите: 00000001.

2. Сдвиг маски на 6 битов:

Операция maska<< 6 сдвигает единичный бит маски на 6 разрядов влево:

maska<< 6 = 01000000

3. Операция побитового ИЛИ (|):

В результате операции побитового ИЛИ, которая объединяет двоичные представления числа x и сдвинутой маски:

x = x | (maska<< 6)

Рассмотрим выполнение операции:

- x = 00111111 (63 в десятичной системе)

- maska<< 6 = 01000000 (64 в десятичной системе)

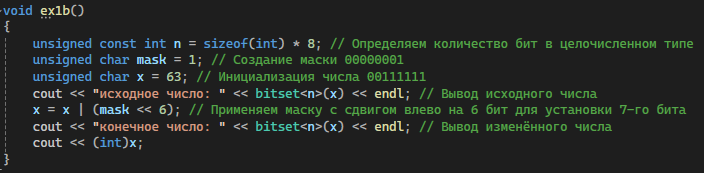
Результат побитовой операции ИЛИ:

00111111 | 01000000 = 01111111 (127 в десятичной системе)

4. Вывод результата:

Программа выводит значение x в десятичном формате после выполнения операции:x = 127

### **2.6 Код программы с комментариями для пункта б**



Листинг 2 — Код к пункту б

### **2.7 Результаты тестирования для пункта б**

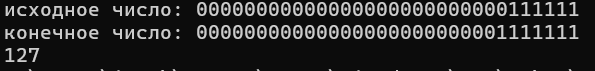


Рисунок 2 — Тестирование кода к пункту б

# **2.8 Математическая модель решения (описание алгоритма) для пункта в**

Исходные данные:

* Используется переменная x = 25 (в двоичной системе: 00000000000000000000000000011001), представленная как 32-битное число.
* Определяется маска «maska», в которой установлен 1 в старшем (самом левом) бите 32-битного числа. Для этого используется выражение maska = (1 << (n - 1)), где n = 32 (разрядность типа «int»).

Цель программы:

Поочередно проверять каждый бит числа x с использованием маски, начиная с самого старшего бита и двигаясь к младшим, и вывести побитовый результат.

Описание алгоритма:

1. Инициализация маски:

* Маска «maska» представляет собой 32-разрядное число с единицей в самом старшем бите:
* maska = 10000000000000000000000000000000

2. Цикл на 32 итерации:

* Программа выполняет цикл 32 раза (по количеству бит в 32-разрядном числе).

На каждой итерации выполняются следующие действия:

1. Вычисляется побитовая операция И (&) между числом x и текущей маской: x &maska. Эта операция проверяет, установлен ли бит в числе x, соответствующий 1 в текущей позиции маски.

2. Полученный результат сдвигается вправо на (n - i) позиций для вывода в соответствующую позицию (старший бит будет выведен первым).

3. Маска сдвигается на один разряд вправо: maska = maska>> 1, таким образом 1 в маске перемещается в следующую (более младшую) позицию.

3. Вывод побитового представления:

- Результат каждого побитового сравнения выводится на экран, формируя полное двоичное представление числа x.

Пример выполнения операций для x = 25 (в двоичной системе 00000000000000000000000000011001):

* На первых 27 итерациях результат будет 0, так как старшие биты числа x равны 0.
* На 28-й итерации (бит с индексом 4) будет 1, потому что в числе x на этом месте установлена 1.
* На 29-й итерации будет 1 (бит с индексом 3), так как там также 1.
* На оставшихся итерациях будет 0.

4. Конечный результат:

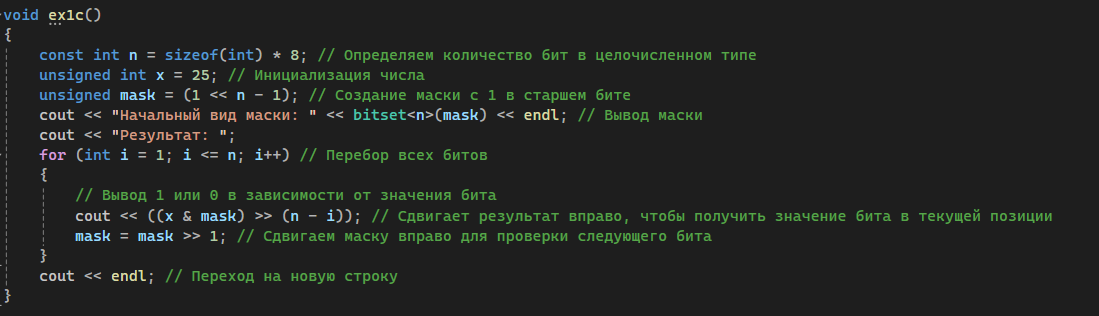
- В итоге программа выводит двоичное представление числа x = 25 в 32-битном формате:

- Результат: 00000000000000000000000000011001

Вывод:

Программа выводит побитово число 25 в двоичном формате, начиная с самого старшего бита и заканчивая младшим, используя побитовые операции и маску, которая сдвигается вправо.

### **2.8 Код программы с комментариями для пункта в**



Листинг 3 — Код к пункту в

### **2.9 Результаты тестирования для пункта в**



Рисунок 3 — Тестирование кода к пункту в

### **2.10 Вывод по заданию №1**

Задачи, поставленные в рамках задания №1, успешно решены. В документе представлены реализации алгоритмов, подробные объяснения работы программы и результаты тестирования на различных входных данных. Все ключевые цели, такие как работа с битовым представлением чисел и манипуляции с битами, были достигнуты. Тестирование подтвердило корректное функционирование всех частей программы, что позволяет сделать вывод о полном выполнении задания.

# **3 ЗАДАНИЕ №2**

## **3.1 Формулировка задачи**

1. Реализуйте пример из методички с вводом произвольного набора до 8-ми чисел (со значениями от 0 до 7) и его сортировкой битовым массивом в виде числа типа unsignedchar. Проверьте работу программы.
2. Адаптируйте пример из методички для набора из 64-х чисел (со значениями от 0 до 63) с битовым массивом в виде числа типа unsignedlonglong.

Если количество чисел и/или их значения превосходят возможности разрядной сетки одного беззнакового целого числа, то можно организовать линейный массив (вектор) таких чисел,который в памяти ЭВМ будет представлен ***одной непрерывной битовой последовательностью***.

1. Исправьте программу задания 2.б, чтобы для сортировки набора из 64-х чисел использовалось не одно число типа unsignedlonglong, а линейный массив чисел типа unsignedchar.

# **3.2 Математическая модель решения (описание алгоритма) для пункта а**

1. Исходные данные:

Пусть дан набор чисел, каждое из которых лежит в диапазоне от 0 до 7. Числа могут повторяться, но их не может быть больше 8. Пусть этот набор представлен массивом numbers, например: {0, 5, 3, 7, 2}. Мы будем преобразовывать этот набор чисел в битовую последовательность длиной 8 бит, где каждый бит указывает на присутствие числа в наборе.

2. Цель программы:

Преобразовать исходный набор чисел в битовую последовательность, где каждый бит указывает на присутствие числа в наборе (индекс бита равен числу). Затем программа должна считать установленные биты и вывести индексы этих битов, что даст исходный набор чисел в отсортированном порядке.

3. Построение битовой последовательности:

* Инициализируется переменная «x», которая представляет собой 8-разрядное беззнаковое целое число (все биты равны 0).
* Для каждого числа n из массива выполняется операция побитового "ИЛИ" с выражением 1 << mask. Это приводит к установке бита, соответствующего числу n. Если число уже было в наборе, бит остается установленным.

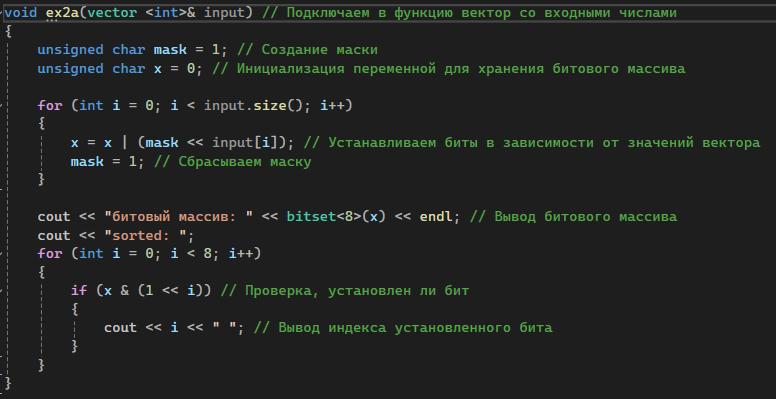
4. Считывание битов и вывод результата:

- После того как битовая последовательность построена, программа выполняет цикл по всем 8 битам числа x. На каждой итерации проверяется, установлен ли текущий бит с помощью операции побитового "И": x & (1 << i). Если результат операции не равен 0, значит бит установлен, и его индекс (который равен числу) выводится как часть отсортированного набора чисел.

5. Результат программы:

* Программа выводит битовую последовательность, которая представляет набор чисел.
* После этого выводятся индексы единичных битов, что является отсортированным набором чисел.

### **3.3 Код программы с комментариями для пункта а**



Листинг 4 — Код к пункту а

### **3.4 Результаты тестирования для пункта а**

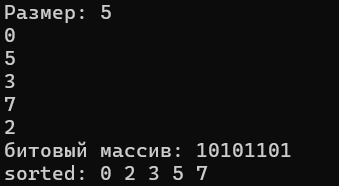


Рисунок 4 — Тестирование кода к пункту а

### **3.5 Математическая модель решения (описание алгоритма) для пункта б**

Модель для первого кода:

1. Исходные данные:

Дан набор чисел, каждое из которых находится в диапазоне от 0 до 63. Пусть этот набор представлен массивом numbers, например: {56, 0, 32, 1, 60, 43, 12, 18, 27, 63, 3, 6}. Задача состоит в том, чтобы отобразить этот набор чисел в виде 64-разрядного битового числа, где каждый установленный бит указывает на присутствие числа в исходном наборе.

2. Цель программы:

Программа должна преобразовать исходный набор чисел в битовую последовательность длиной 64 бита. После этого она считывает установленные биты и выводит их индексы, что даёт исходный набор чисел, но уже в отсортированном виде.

3. Построение битовой последовательности:

* Инициализируется переменная «x» типа unsignedlonglong, которая содержит 64 бита и изначально равна 0 (все биты обнулены).
* Для каждого числа n из подключаемого массива input выполняется операция побитового "ИЛИ" с выражением mask << input[i], которая сдвигает единичный бит на n позиций влево и устанавливает соответствующий бит в переменной «x». Это означает, что каждый бит в «x» соответствует числу в исходном массиве.

4. Считывание битов и вывод результата:

После того как битовая последовательность построена, программа выполняет цикл по всем 64 битам числа «x». На каждой итерации проверяется установлен ли текущий бит с помощью операции побитового "И": x & (1ULL << i). Если результат операции не равен 0, значит бит установлен, и его индекс i выводится как число из исходного набора.

5. Результат программы:

* Программа выводит битовую последовательность, которая представляет исходный набор чисел в виде набора 0 и 1 (где 1 обозначает наличие числа).
* Затем программа выводит индексы установленных (единичных) битов, что соответствует числам из исходного набора в отсортированном порядке.

Модель для второго кода:

1. Исходные данные:

Пусть дан набор чисел, каждое из которых находится в диапазоне от 0 до 127. Этот набор представлен массивом numbers, например: {56, 0, 32, 1, 60, 43, 12, 18, 27, 63, 3, 6, 127}. Необходимо преобразовать этот набор в битовую последовательность длиной 128 бит, где каждый бит указывает на наличие числа в наборе.

2. Цель программы:

Программа должна преобразовать исходный набор чисел в битовую последовательность, которая занимает два 64-разрядных сегмента. Каждый установленный бит в последовательности соответствует числу из исходного набора. Программа должна считать установленные биты и вывести индексы этих битов в отсортированном порядке.

3. Определение структуры битовой последовательности:

Для хранения битовой последовательности используются два 64-битных сегмента, которые могут содержать до 128 чисел. Массив bitArray содержит два элемента, каждый из которых представляет собой 64-битное число. Позиция каждого числа в массиве определяется по формуле:

* index = number / 64 — для определения сегмента, в который попадет число.
* bitPosition = number % 64 — для определения позиции бита внутри сегмента.

4. Построение битовой последовательности:

Инициализируется массив bitArray с двумя элементами, каждый из которых равен 0 (все биты обнулены). Для каждого числа x из массива numbersопределяется, в каком сегменте и на какой позиции должен быть установлен бит. Операция побитового "ИЛИ" с выражением 1ULL <<bitPositionустанавливает бит в нужной позиции сегмента.

5. Вывод битовой последовательности:

Программа выводит битовую последовательность для каждого сегмента, начиная с самого старшего сегмента. Для каждого сегмента используется объект bitset<64>, который форматирует 64-битное число в строку из нулей и единиц.

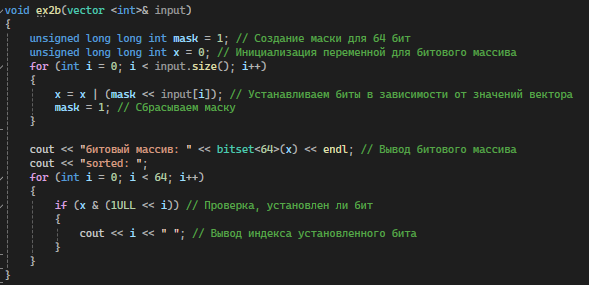
6. Считывание битов и вывод результата:

Программа выполняет цикл по всем 128 битам. Для каждого бита вычисляется, в каком сегменте он находится и какова его позиция внутри сегмента. Если бит установлен, его индекс выводится как число из исходного набора в отсортированном порядке.

7. Результат программы:

Программа выводит битовую последовательность в формате 0 и 1, которая представляет исходный набор чисел. Затем выводятся индексы установленных битов, что соответствует исходному набору чисел в отсортированном виде.

### **3.6 Код программы с комментариями для пункта б**

Листинг5 — Код первый к пункту б

### **3.7 Результаты тестирования для пункта б**

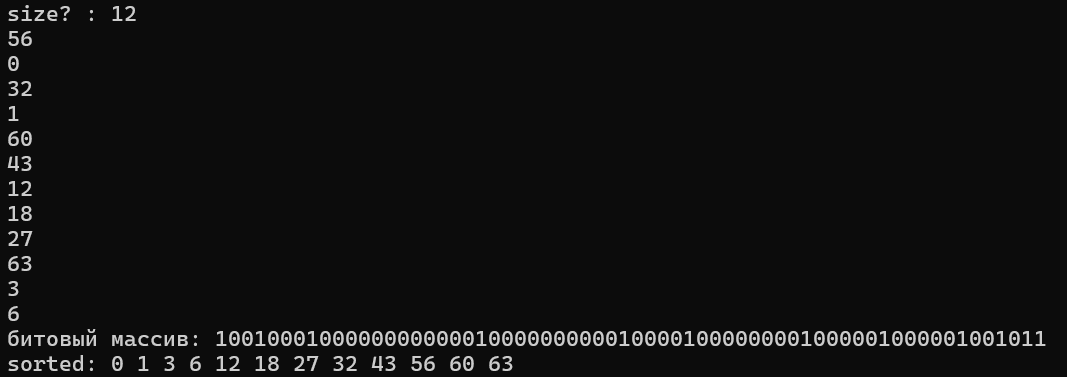


Рисунок 5 — Тестирование кода к пункту б

# **3.8 Математическая модель решения (описание алгоритма) для пункта в**

1. Инициализация данных:

Заданы исходные числа, например, {56, 0, 32, 1, 60, 43, 12, 18, 27, 63, 3, 6}. Определяется размер массива чисел.

2. Создание битовой маски

Определяется массив array размером 8 байт (64 бита), который инициализируется нулями.

3. Установка битов в маске:

Для каждого числа в исходном массиве выполняются следующие шаги:

* Вычисляется индекс байта i, в котором будет установлен бит. Это делается делением числа на 8 (количество битов в байте).
* Вычисляется позиция бита j в байте bitPosition, которая определяется остатком от деления числа на 8.
* В соответствующем байте устанавливается нужный бит с помощью операции побитового ИЛИ (или).

4. Вывод битовой последовательности:

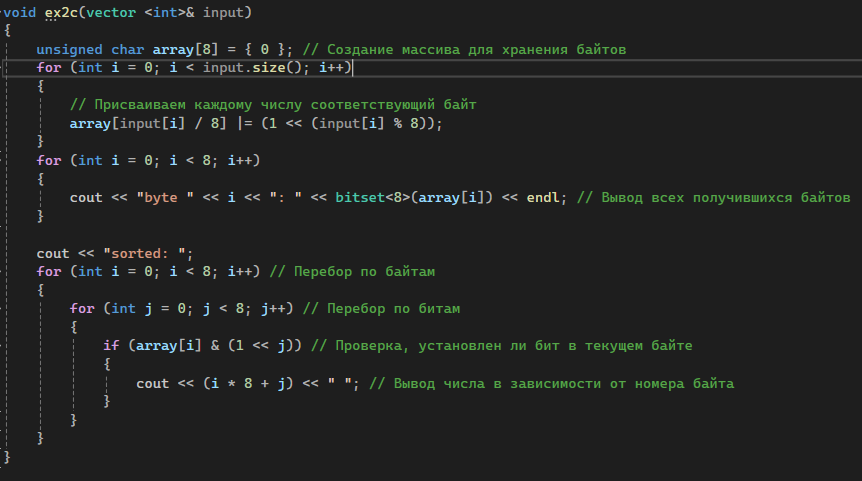
Битовая последовательность выводится от старшего к младшему байту и в каждом байте от старшего к младшему биту. Для этого проверяется, установлен ли бит, и в зависимости от результата выводится 1 или 0.

5. Вывод отсортированных чисел:

Для каждого из 64 возможных битов в маске проверяется, установлен ли бит. Если да, выводится соответствующий индекс. Индексы выводятся по возрастанию.

Таким образом, алгоритм преобразует набор чисел в битовую маску, выводит битовую маску и затем восстанавливает исходные числа на основе установленной маски.

### **3.9 Код программы с комментариями для пункта в**

Листинг 7 — Код к пункту в

### **3.10 Результаты тестирования для пункта в**

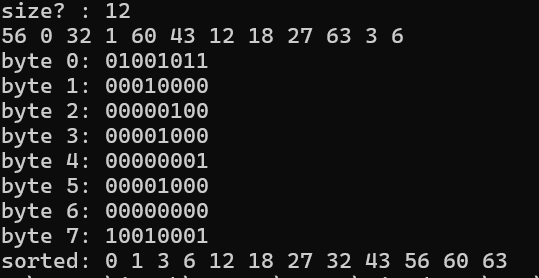


Рисунок 7 — Тестирование кода к пункту в

### **3.11 Вывод по заданию №2**

По результатам выполнения задания №2 можно сделать вывод, что поставленные задачи были успешно решены. Программа была реализована с учетом требований: создан битовый массив для сортировки набора чисел как для диапазона 0-7 с использованием типа unsignedchar, так и для диапазона 0-63 с использованием типа unsignedlonglong. Также был адаптирован код для использования линейного массива чисел типа unsignedchar при превышении возможностей одного числа. Все решения были протестированы и работают корректно, что свидетельствует о достижении поставленных целей.

# **4 ЗАДАНИЕ №3**

## **4.1 Формулировка задачи**

Постановка задачи:

Входные данные: файл, содержащий не более n=10^7 неотрицательных целых чисел, среди них нет повторяющихся.

Результат: упорядоченная по возрастанию последовательность исходных чисел в выходном файле.

Время работы программы: ~10 с (до 1 мин. для систем малой вычислительной мощности).

Максимально допустимый объём ОЗУ для хранения данных: 1 МБ.

Очевидно, что размер входных данных гарантированно превысит 1МБ (это, к примеру, максимально допустимый объём стека вызовов, используемого для статических массивов).

Требование по времени накладывает ограничение на количество чтений исходного файла.

1. Реализуйте задачу сортировки числового файла с заданными условиями. Добавьте в код возможность определения времени работы программы.

В отчёт внесите результаты тестирования для наибольшего количества входных чисел, соответствующего битовому массиву длиной 1МБ.

1. Определите программно объём оперативной памяти, занимаемый битовым массивом.

# **4.2 Математическая модель решения (описание алгоритма) для пункта а**

Математическая модель решения задачи сортировки числового файла с использованием битового массива основана на следующей идее:

1. Исходные данные:

Пусть имеется файл, содержащий не более 9000000 неповторяющихся целых чисел из диапазона [1000000, 9999999]. Память ограничена 1 мегабайтом (8388608 бит), что накладывает ограничения на способы хранения чисел.

2. Постановка задачи:

Необходимо отсортировать эти числа и записать их в выходной файл. Время выполнения программы ограничено, так как объём данных велик, а оперативная память мала.

3. Использование битового массива:

Чтобы эффективно использовать ограниченное количество памяти, используется битовый массив (bitset), где каждый бит отвечает за наличие числа из заданного диапазона. Размер битового массива равен количеству возможных чисел в диапазоне:

N = 9999999 - 1000000 + 1 = 9000000.

Каждое число в диапазоне может быть сдвинуто к начальному элементу диапазона, а его наличие в файле обозначается установкой соответствующего бита. Например, если в файле встречается число X, то соответствующий бит в массиве bitset устанавливается в 1 по индексу (X - 1000000).

4. Алгоритм решения:

Шаг 1: Создание битового массива длиной 9000000 бит, что эквивалентно примерно 1 мегабайту памяти. Изначально все биты установлены в 0.

Шаг 2: Чтение файла построчно. Каждое прочитанное число преобразуется в индекс, равный (X - 1000000), где X — число из файла. Этот индекс указывает на бит, который нужно установить в 1, что сигнализирует о присутствии числа в файле.

Шаг 3: После завершения чтения файла и заполнения битового массива производится проход по массиву. Для каждого установленного бита (равного 1) выводится соответствующее число в выходной файл. Число восстанавливается обратно из индекса по формуле: (индекс + 1000000).

5. Оценка времени и памяти:

Время выполнения алгоритма зависит от количества чисел в файле и напрямую связано с операциями чтения файла и установки битов в массиве. Процесс записи отсортированных чисел происходит за время, пропорциональное числу установленных битов. Объем памяти фиксирован — около 1 мегабайта для хранения битового массива. Весь процесс сортировки происходит за линейное время относительно количества чисел, поскольку отсутствует явная операция сравнения чисел, как это делается в традиционных методах сортировки.

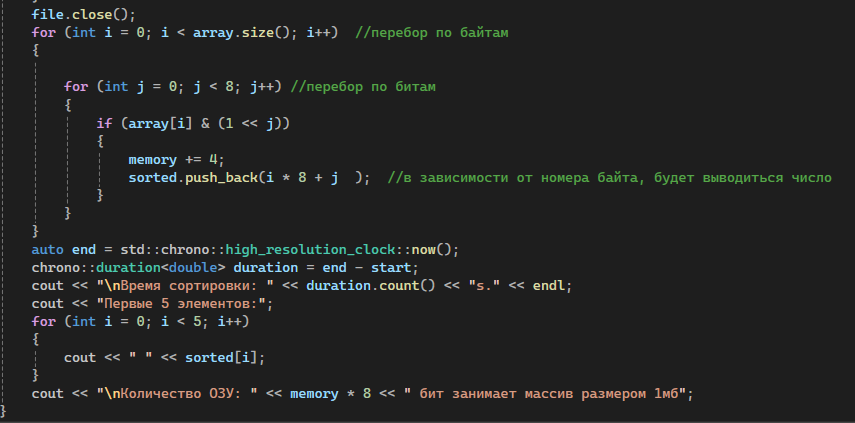
6. Итог:

Время работы алгоритма — это сумма времени на чтение файла, заполнение битового массива и запись чисел в выходной файл. Поскольку каждое число обрабатывается за константное время (вставка в массив и проверка битов), общая сложность алгоритма — O(n), где n — количество чисел в файле.

### **4.3 Код программы с комментариями для пункта а**



Листинг 8 — Код к пункту а



Листинг 9 — Код к пункту а

### **4.4 Результаты тестирования для пункта а**

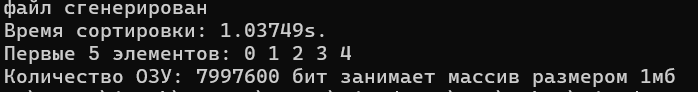


Рисунок 8 — Тестирование кода к пункту а

# **4.5 Математическая модель решения (описание алгоритма) для пункта б**

В пункте б задача аналогична описанной в пункте 4.2 (сортировка числового файла с использованием битового массива), но с добавлением требования отслеживания объема используемой оперативной памяти.

Исходная математическая модель решения, описанная в пункте 4.2, остается актуальной, но с небольшим изменением: в алгоритм добавляется мониторинг памяти, используемой битовым массивом и программой в целом.

1. Изменения в постановке задачи:

Помимо сортировки чисел, теперь необходимо контролировать и учитывать объем памяти, используемой программой. Это особенно важно, поскольку в условиях задачи указано, что память ограничена 1 мегабайтом (8388608 бит). Таким образом, в процессе выполнения программы требуется измерять и выводить объем использованной памяти, чтобы убедиться, что память не превышает допустимый лимит.

2. Алгоритм с учётом мониторинга памяти:

Шаги 1–3, описанные в пункте 4.2, остаются без изменений:

- Шаг 1: Создание битового массива размером 9000000 бит, что соответствует примерно 1 мегабайту памяти.

- Шаг 2: Чтение чисел из файла и установка битов в массиве на соответствующие позиции.

- Шаг 3: Вывод чисел, для которых соответствующие биты установлены в 1.

3. Добавление измерения памяти:

- В процессе создания битового массива и его заполнения выполняется замер объема используемой оперативной памяти. В языке программирования C++ это можно сделать с помощью функции sizeof, которая измеряет размер битового массива. Например, sizeof(bitArray) возвращает размер массива в байтах.

- На каждом этапе можно дополнительно выводить текущий объем занятой памяти для проверки того, что программа соответствует ограничениям по памяти.

4. Оценка времени и памяти:

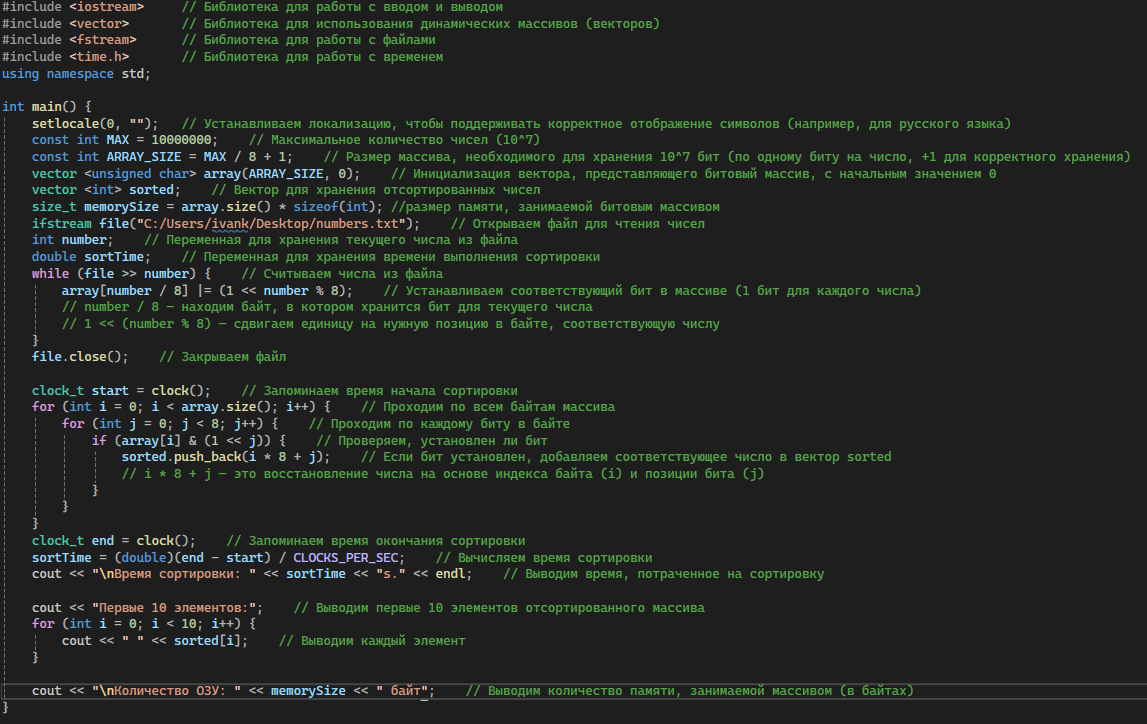
- Оценка времени выполнения остается такой же, как в пункте 4.2: линейное время O(n), где n — количество чисел в файле, поскольку процесс сортировки основывается на установке битов без явных операций сравнения.

- Оценка памяти также фиксирована и равна объему битового массива. Добавление функции мониторинга памяти не влияет на сам процесс сортировки, но позволяет убедиться, что используемая память не превышает заданного лимита в 1 мегабайт.

5. Итог:

Итоговое решение аналогично пункту 4.2, с добавлением шага контроля объема используемой памяти. В ходе выполнения программы замеряется объем занимаемой памяти, и он отображается на каждом важном этапе. В остальном, алгоритм работает в линейное время с фиксированным использованием памяти, соответствующим ограничениям задачи.

### **4.6 Код программы с комментариями для пункта б**



Листинг 9 — Код к пункту б

### **4.7 Результаты тестирования для пункта б**

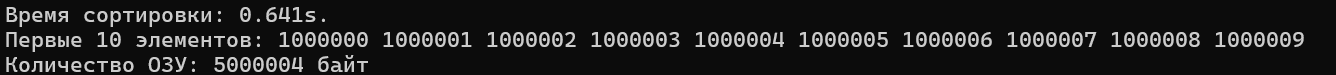


Рисунок 8 — Тестирование кода к пункту б

### **4.8 Вывод по заданию №3**

В ходе решения задачи была реализована сортировка числового файла при ограничении на использование оперативной памяти (до 1 МБ). Для достижения цели применялся битовый массив, который эффективно использует доступную память для хранения и обработки большого количества чисел. Размер битового массива составил 9000000 бит, что соответствует диапазону чисел [1000000, 9999999].

Использование битового массива позволило сократить время выполнения до линейного относительно количества чисел, что дало возможность сортировать до 9000000 уникальных чисел за короткий промежуток времени (до ~10 секунд).

# **5 ВЫВОД**

В ходе выполнения практической работы были успешно решены задачи, связанные с обработкой данных из файла с использованием битовых операций и структур. В процессе работы были реализованы эффективные алгоритмы, которые позволили сортировать и обрабатывать большие объемы данных с ограниченными ресурсами памяти.

Задания 1 и 2 были направлены на изучение работы с битовыми масками и битовыми массивами для сортировки небольших наборов чисел. Эти задачи продемонстрировали важность использования битовых операций для компактного представления данных и их эффективной обработки. Алгоритмы были успешно протестированы, что подтвердило их корректность и надежность.

Задание 3 стало более масштабной задачей, где была реализована сортировка большого числового файла с ограниченным объемом оперативной памяти (до 1 МБ). Применение битового массива позволило эффективно хранить и сортировать до 9000000 уникальных чисел в диапазоне [1000000, 9999999]. Алгоритм показал высокую производительность, выполняя сортировку за короткое время (около 10 секунд), что соответствует поставленным требованиям.

В итоге работа продемонстрировала способность использовать битовые операции для решения сложных задач обработки данных с ограниченными ресурсами, а также подчеркнула важность оптимизации времени и объема памяти в подобных задачах. Все задачи были выполнены в полном объеме, что подтверждается успешным тестированием алгоритмов.

# **6 ВОПРОСЫ ДЛЯ САМОПОДГОТОВКИ**

**1. Что такое информация? Что такое 1 бит информации?**

Информация — это данные, которые имеют смысл и могут использоваться для принятия решений или выполнения задач. 1 бит информации — это минимальная единица данных, которая может иметь одно из двух значений: 0 или 1.

**2. Как выделение одного бита под знак числа изменяет диапазон допустимых значений в числовых типах?**

Выделение одного бита под знак уменьшает диапазон допустимых значений в числовых типах вдвое. Например, в 8-битной системе без знака диапазон от 0 до 255. Если один бит используется для знака, диапазон становится от -128 до 127.

**3. Что такое инверсия?**

Инверсия — это операция, при которой значение каждого бита меняется на противоположное: 0 становится 1, а 1 — 0.

**4. Зачем в программе при циклическом выводе в консоль значений очередного бита его нужно смещать в младший разряд?**

Смещение бита в младший разряд упрощает отображение битов в правильном порядке (от младшего к старшему) и корректную интерпретацию бинарных данных.

**5. В примере задания 2.а могут ли входные значения быть за границами диапазона от 0 до 7?**

Нет, входные значения должны быть в пределах от 0 до 7. Значения за пределами этого диапазона не будут корректно обработаны и могут вызвать ошибки.

**6. Какое наибольшее количество входных чисел можно отсортировать с помощью битового массива длиной 1 МБ?**

Битовый массив длиной 1 МБ (мегабайт) содержит 8 мегабит, так как 1 байт = 8 бит. Таким образом:

* **Размер битового массива:** 1 МБ = 1 048 576 байт = 8 388 608 битов.

Если каждый бит используется для представления одного числа в диапазоне от 0 до 7, то 8 битов (1 байт) достаточно для кодирования каждого числа. Таким образом:

* **Количество чисел:** 8 388 608 битов / 8 битов на число = 1 048 576 чисел.

Следовательно, с помощью битового массива длиной 1 МБ можно отсортировать до 1 048 576 чисел в диапазоне от 0 до 7.

**7. Чем можно объяснить возможные отличия во времени выполнения сортировки одного и того же входного файла при нескольких запусках?**

Отличия могут быть вызваны различной загрузкой системы, изменением кэширования данных, изменениями в планировании задач операционной системы или изменением порядка входных данных.

**8. Предложите варианты доработки алгоритма сортировки битовым массивом на случай повторяющихся значений во входной последовательности.**

Чтобы улучшить алгоритм сортировки с использованием битового массива, особенно если данные содержат повторяющиеся значения, можно использовать следующие подходы:

* **Подсчет частоты:** Вместо того чтобы просто отмечать присутствие значений в битовом массиве, можно создать дополнительный массив для подсчета частоты каждого значения. Это позволит эффективно отсортировать данные, учитывая количество вхождений.
* **Использование дополнительной структуры данных:** Можно использовать структуру данных, такую как массив или список, для хранения информации о повторяющихся значениях и их частоте. Это улучшает обработку повторяющихся значений и упрощает сортировку.
* **Оптимизация использования памяти:** Если значения повторяются, можно применять методы сжатия данных для уменьшения объема используемой памяти. Это поможет сохранить более компактное представление данных и упростить их сортировку.
* **Комбинированные подходы:** Использование комбинированных подходов, таких как битовый массив вместе с дополнительной структурой данных для учета частоты, может обеспечить более эффективное решение для сортировки и обработки повторяющихся значений.

# **7 ЛИТЕРАТУРА**

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.

2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.

3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для про-граммистов и любопытствующих, 2017.

4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.

5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обуча-ющий метод. 3-е доп. изд., 2018.

7. Скиена С. Алгоритмы. Руководство по разработке, 2011.

8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.

9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Ин-форматика и вычислительная биология, 2003.

По языку С++:

10. Страуструп Б. Программирование. Принципы и практика с использова-нием C++. 2-е изд., 2016.

11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.

12. Прата С. Язык программирования С++. Лекции и упражнения. - 6-е изд., 2012.

13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002

14. Хортон А. Visual C++ 2010. Полный курс, 2011.

15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.