



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего  
образования*

*«МИРЭА – Российский технологический университет»*

**РТУ МИРЭА**

---

---

Отчет практической работе №5.1

**Тема:**

**«Битовые операции. Сортировка числового файла с помощью битового  
массива»**

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнила студент группы ИКБО-42-23

Туляшева А.Т.

Принял ассистент

Муравьева Е.А.

Москва 2024

## Содержание

<b>ЦЕЛЬ РАБОТЫ</b> .....	3
<b>ЗАДАНИЕ 1</b> .....	3
1.1 Упражнение 1.....	3
1.2 Упражнение 2.....	4
1.3 Упражнение 3 .....	5
1.4 Упражнение 4 .....	6
1.5 Упражнение 5 .....	6
<b>ЗАДАНИЕ 2</b> .....	8
2. 1 Задание 2.....	8
<b>ЗАДАНИЕ 3</b> .....	10
3.1 Задание 3.....	10
<b>ВЫВОД</b> .....	13

## ЦЕЛЬ РАБОТЫ

Освоить приемы работы с битовым представлением беззнаковых целых чисел, реализовать эффективный алгоритм внешней сортировки на основе битового массива.

## ЗАДАНИЕ 1

### 1.1 Упражнение 1.

#### Формулировка упражнения:

Определить переменную целого типа, присвоить ей значение, используя константу в шестнадцатеричной системе счисления.

Разработать оператор присваивания и его выражение, которое установит пять младших битов исходного значения переменной в значение 1, используя соответствующую маску и поразрядную операцию.

#### Математическая модель решения (описание алгоритма):

Рассмотрим на примере. Дано:

- $x = 152$  (в двоичном виде: 10011000)
- $maska = 31$  (в двоичном виде: 00011111)

Алгоритм:

1. Применяем побитовое ИЛИ между  $x$  и маской.

$$x \mid maska = 10011000 \mid 00011111 = 10011111$$

Результат:  $x$  изменяется со 152 на 159. Алгоритм использует битовые операции для установления пяти младших битов числа  $x$  в значение 1.

#### Код программы

```
void bit_1() {  
    int x = 0x98;  
    int maska = 0x1F;  
    cout << x << " ";  
    cout << bitset<16>(x) << endl;  
    x = x | maska;  
    cout << x << " ";  
    cout << bitset<16>(x) << endl;  
}
```

Рисунок 1 – Код упражнения 1

## Результаты тестирования

Проведем тестирование:

```
Исходное число: 152      Исходное число в двоичном виде: 0000000010011000
Итоговое число: 159      Итоговое число в двоичном виде: 0000000010011111
Пять младших битов установлены в значение 1
```

Рисунок 2 – Тестирование упражнения 1

Тестирование показало, что программа работает корректно.

### 1.2 Упражнение 2.

#### Формулировка упражнения:

Определить переменную целого типа.

Разработать оператор присваивания и его выражение, которое обнуляет 7-й, 9-ый, 5-ый биты исходного значения переменной, используя соответствующую маску и поразрядную операцию. Значение в переменную вводится с клавиатуры.

#### Математическая модель решения (описание алгоритма):

Алгоритм:

1. Применяем побитовое И между x и маской.

Алгоритм использует битовые операции для установки в 0 7-го, 9-го и 5-го битов числа x.

#### Код программы

```
void bit_2() {
    int x; //FDF, 4063
    cin >> x;
    int maska = 0xEAF;
    cout << x << " ";
    cout << bitset<16>(x) << endl;
    x = x & maska;
    cout << x << " ";
    cout << bitset<16>(x) << endl;
}
```

Рисунок 3 – Код упражнения 2

## Результаты тестирования

Проведем тестирование:

```
Введите число: 4063
Исходное число: 4063      Исходное число в двоичном виде: 0000111111011111
Итоговое число: 3727      Итоговое число в двоичном виде: 0000111010001111
5-ой, 7-ый, 9-ый биты были обнулены
```

Рисунок 4 – Тестирование упражнения 2

Тестирование показало, что программа работает корректно.

### 1.3 Упражнение 3

#### Формулировка упражнения:

Определить переменную целого типа.

Разработать оператор присваивания и выражение, которое умножает значение переменной на 64, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.

#### Математическая модель решения (описание алгоритма):

Алгоритм:

1. Сдвигаем x влево на 6 позиций

Алгоритм использует битовые операции для умножения значения переменной на 64.

#### Код программы

```
void bit_3() {
    int x;
    cin >> x;
    int number = 64;
    cout << x << " ";
    cout << bitset<16>(x) << endl;
    x = x << 6;
    cout << x << " ";
    cout << bitset<16>(x) << endl;
}
```

Рисунок 5 – Код упражнения 3

#### Результаты тестирования

Проведем тестирование:

```
Введите число: 2
Исходное число: 2      Исходное число в двоичном виде: 0000000000000010
Итоговое число: 128    Итоговое число в двоичном виде: 0000000010000000
Число было умножено на 64
```

Рисунок 6 – Тестирование упражнения 3

Тестирование показало, что программа работает корректно.

## 1.4 Упражнение 4

### Формулировка упражнения:

Определить переменную целого типа.

Разработать оператор присваивания и выражение, которое делит значение переменной на 64, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.

### Математическая модель решения (описание алгоритма):

Алгоритм:

1. Сдвигаем x вправо на 6 позиций

Алгоритм использует битовые операции для деления значения переменной на 64.

### Код программы

```
void bit_4() {  
    int x;  
    cin >> x;  
    int number = 64;  
    cout << x << " ";  
    cout << bitset<16>(x) << endl;  
    x = x >> 6;  
    cout << x << " ";  
    cout << bitset<16>(x) << endl;  
}
```

Рисунок 7 – Код упражнения 4

### Результаты тестирования

Проведем тестирование:

```
Введите число: 64  
Исходное число: 64      Исходное число в двоичном виде: 0000000001000000  
Итоговое число: 1      Итоговое число в двоичном виде: 0000000000000001  
Число было разделено на 64
```

Рисунок 8 – Тестирование упражнения 4

Тестирование показало, что программа работает корректно.

## 1.5 Упражнение 5

### Формулировка упражнения:

Определить переменную целого типа.

Разработать оператор присваивания и выражение, в котором используются только поразрядные операции. В выражении используется маска – переменная. Маска инициализирована единицей в старшем разряде (вар 2). Необходимо установить n-ый бит в 1. Изменяемое число вводится с клавиатуры.

### Математическая модель решения (описание алгоритма):

Алгоритм:

1. Введем маску, которая инициализирована единицей в старшем разряде, и бит, который хотим установить в значение 1.
2. Изменим маску, сдвинув ее значение вправо до нужного бита (16-n).
3. Применим побитовое ИЛИ между x и маской.

Алгоритм использует битовые операции для деления значения переменной на 64.

### Код программы

```
void bit_5() {  
    int x;  
    cin >> x;  
    int maska = 0x8000;  
    int n;  
    cin >> n;  
    maska = maska >> (16 - n);  
    cout << x << " ";  
    cout << bitset<16>(x) << endl;  
    x = x | maska;  
    cout << x << " ";  
    cout << bitset<16>(x) << endl;  
}
```

Рисунок 9 – Код упражнения 5

### Результаты тестирования

Проведем тестирование:

```
Введите число: 0  
Введите бит, который хотите установить в 1 (от 1 до 16): 14  
Исходное число: 0          Исходное число в двоичном виде: 0000000000000000  
Итоговое число: 8192      Итоговое число в двоичном виде: 0010000000000000  
14-й бит был установлен в 1
```

Рисунок 10 – Тестирование упражнения 5

Тестирование показало, что программа работает корректно.

## ЗАДАНИЕ 2

### 2.1 Задание 2.

#### Формулировка задачи:

а) Реализуйте вышеописанный пример с вводом произвольного набора до 8-ми чисел (со значениями от 0 до 7) и его сортировкой битовым массивом в виде числа типа unsigned char. Проверьте работу программы.

б) Исправьте программу задания, чтобы для сортировки набора из 64-х чисел использовалось не одно число типа unsigned long long, а линейный массив чисел типа unsigned char.

#### Математическая модель

1. Введем число, означающее размер массива и сами числа массива. Инициализируем маску единицей в старшем разряде.

2. Набор чисел отражаем в виде 8-разрядной битовой последовательности, где единичные биты показывают наличие в исходном наборе числа, равного номеру этого бита в последовательности.

3. Последовательное считывание бит этой последовательности и вывод индексов единичных битов позволяет получить исходный набор чисел в отсортированном виде.

#### Код программы

```
void bit_sort() {
    int n;
    cout << "Введите количество чисел:\t";
    cin >> n;
    int x;
    unsigned char maska = 0x80;
    unsigned char arr = 0;
    cout << "Введите массив чисел:\t";
    for (int i = 0; i < n; i++) {
        cin >> x;
        arr = arr | (maska >> x); //11010111
    }
    cout << "Отсортированный массив:\t";
    for (int i = 0; i < 8; i++) {
        if (arr == (arr | maska)) {
            cout << i << " ";
        }
        maska >>= 1;
    }
}
```

Рисунок 11 – Код задания 2а



## Результаты тестирования

Проведем тестирование на различных вводных данных:

```
Введите количество чисел (до 8): 8
Введите массив чисел (от 0 до 7): 7 1 0 2 6 4 3 5
Отсортированный массив: 0 1 2 3 4 5 6 7
```

Рисунок 12 - Тестирование кода задания 2а

Тестирование показало, что программа работает корректно.

## Код программы

```
void bit_sort2() {
    int n;
    cin >> n;
    int x;
    unsigned char maska = 0x80;
    vector<unsigned char> arr = { 0, 0, 0, 0, 0, 0, 0, 0 };
    for (int i = 0; i < n; i++) {
        cin >> x;
        arr[x / 8] = arr[x / 8] | (maska >> x % 8); //00000000 00100000
    }
    for (int i = 0; i < 64; i++) {
        if (arr[i / 8] == (arr[i / 8] | maska)) {
            cout << i << " ";
        }
        if ((i + 1) % 8 == 0) {
            maska <<= 7;
        } else maska >>= 1;
    }
}
```

Рисунок 13 – Код задания 2б

## Результаты тестирования

Проведем тестирование на различных вводных данных:

```
Введите количество чисел (до 64): 10
Введите массив чисел (от 0 до 63): 62 10 2 8 1 6 22 31 9 3
Отсортированный массив: 1 2 3 6 8 9 10 22 31 62
```

Рисунок 14 – Тестирование кода задания 2б

Тестирование показало, что программа работает корректно.

## **ЗАДАНИЕ 3**

### **3.1 Задание 3.**

#### **Формулировка 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:  $\sim 10$  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.

Реализуйте тестовый пример, демонстрирующий входные данные и заполненный битовый массив (не более 20 чисел).

Реализуйте задачу сортировки числового файла для входных данных объемом 100 и 1000 чисел. Показать время выполнения сортировки для каждого объема.

Реализуйте задачу сортировки заданного числового файла.

#### **Код программы**

```

void bit_sort3() {
    int NUM9 = 10000000; //макс число
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    vector<unsigned char> arr;
    unsigned char maska = 1;

    time_t begin = clock();
    ifstream file("C:\\Users\\arina\\OneDrive\\Рабочий стол\\мои файлы\\дз\\doc1.txt");
    ofstream file_out("C:\\Users\\arina\\OneDrive\\Рабочий стол\\мои файлы\\дз\\doc2.txt");

    //ifstream file("C:\\Users\\arina\\OneDrive\\Рабочий стол\\мои файлы\\дз\\doc1.txt");
    //ofstream file_out("C:\\Users\\arina\\OneDrive\\Рабочий стол\\мои файлы\\дз\\doc2.txt");
    for (int i = 0; i < NUM9 / 8 + 1; i++) arr.push_back(0);
    int q;
    while (file >> q) arr[q / 8] = arr[q / 8] | maska << (q % 8);
    for (int i = 0; i < NUM9 / 8 + 1; i++) {
        for (int j = 0; j < 8; j++) {
            q = (arr[i] & maska) >> j;
            if (q == 1) {
                file_out << i * 8 + j << " ";
            }
            maska <<= 1;
        }
        maska = 1;
    }
    file_out.close();
    time_t end = clock();
    cout << "Время выполнения программы: " << (double)(end - begin) / CLOCKS_PER_SEC;
    cout << "\nОбъем памяти для битового массива: " << sizeof(unsigned long long) * ((NUM9 + 63) / 8) << " байт" << endl;
    //макс число + макс сдвиг в ряду из 64 цифр / 8 для байт
}

```

Рисунок 15 – Код задания 3

## Результаты тестирования

Проведем тестирование на разном количестве чисел:

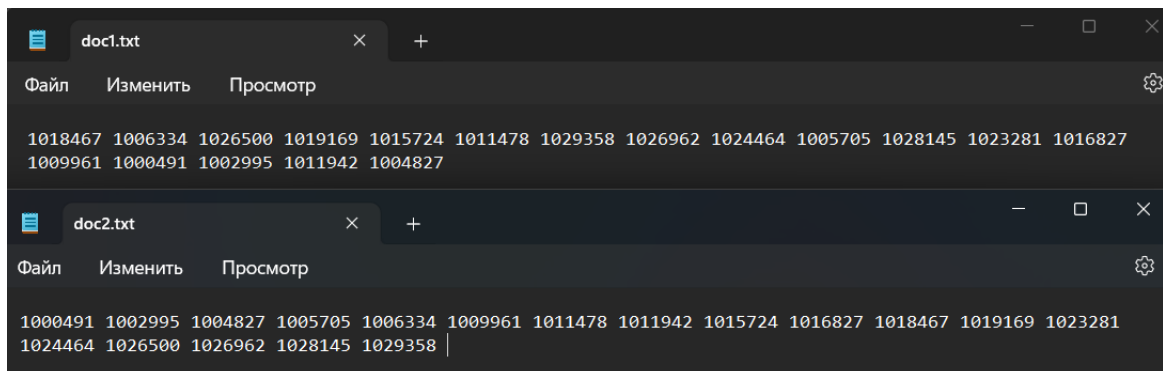


Рисунок 16 – Тестирование на 18 числах

```

Время выполнения программы: 0.09
Объем памяти для битового массива: 4104 байт

```

Рисунок 17 – Тестирование на 100 числах

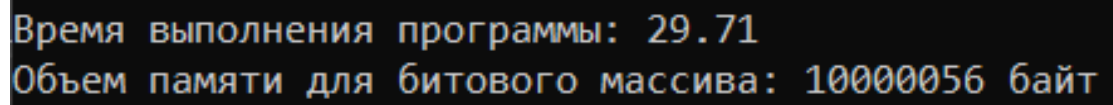
```

Время выполнения программы: 0.102
Объем памяти для битового массива: 4104 байт

```

Рисунок 18 – Тестирование на 1000 числах

Проведем тестирование на всем диапазоне чисел, не превышающих  $10^7$ :



Время выполнения программы: 29.71  
Объем памяти для битового массива: 10000056 байт

Рисунок 19 – Тестирование для наибольшего количества входных чисел

## **ВЫВОД**

В ходе выполнения практической работы была успешно решена задача сортировки чисел с использованием битового массива. Реализован алгоритм, который позволяет считывать до 10 миллионов неотрицательных целых чисел из файла, устанавливать соответствующие биты в битовом массиве и формировать упорядоченный выходной файл.

Цель работы была достигнута: освоены приемы работы с битами и реализован алгоритм, демонстрирующий высокую производительность при сортировке больших объемов данных.