



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1
Сортировка числового файла с помощью битового массива
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-01-21

Луковников Д.Р.

Принял преподаватель

Сартаков М.В.

Практическая

«__»_____2022 г.

работа выполнена

«Зачтено»

«__»_____2022 г.

Москва 2022

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
1 ЦЕЛЬ РАБОТЫ	3
2 ХОД РАБОТЫ	4
2.1 Задание 1.а.....	4
2.2 Задание 1.б.....	5
2.3 Задание 1.в.....	5
2.4 Задание 2.а.....	6
2.5 Задание 2.б.....	7
2.6 Задание 2.в.....	8
2.7 Задание 3.а и 3.б.....	8
ВЫВОДЫ	11
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	12

1 ЦЕЛЬ РАБОТЫ

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

2 ХОД РАБОТЫ

2.1 Задание 1.а

Формулировка задачи: установить 5-й бит произвольного целого числа в 0. Проверить работоспособность программы.

Математическая модель решения: Мы используем маску, в которой изначально содержится одна единица на самой правой позиции. Для того, чтобы установить 5-й бит в положения 0, с помощью побитового сдвига, перемещаем единицу маски на нужную позицию. Затем, чтобы занулить только 1 бит, инвертируем маску (получаем все единицы, кроме 5-й позиции) и производим операцию поразрядной конъюнкции. Все биты, кроме 5-го не изменятся, а 5-й в независимости от его значения станет равен 0.

Листинг 1.1 – Код программы

```
void task1a() {  
    cout << "Task 1.a\n";  
    unsigned char x = 255; // 255 16 (00010000)  
    unsigned char mask = 1;  
    x = x & (~ (mask << 4)); // Shift bits, invert and conjunction  
    cout << bitset<8>(x); // Outputting bit values  
}
```

Тестирование: для начала возьмём число 255, в двоичном представлении оно равно 11111111, в результате работы программы получаем следующее (Рис. 1).

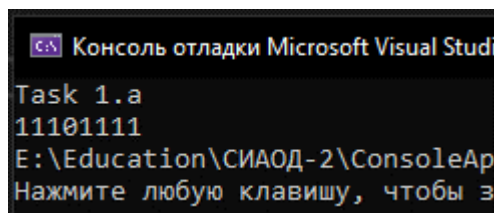


Рисунок 1 – Тестирование для входного числа 255

Выходное значение соответствует ожидаемому результату, 5-й бит занулился. Теперь проверим для другого числа, например - 16, в двоичном представлении оно равно 10000, т.е. оно имеет одну единицу на 5-й позиции. Результат на рисунке 2.

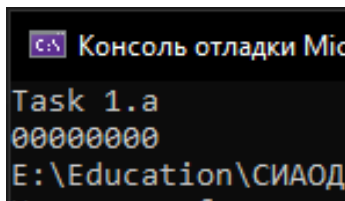


Рисунок 2 – Тестирование для входного числа 16

Бит занулился и тем самым само число так же обнулилось.

Вывод: Данный подход к изменению конкретного бита работоспособен и просто в реализации.

2.2 Задание 1.б

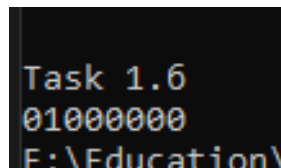
Формулировка задачи: установить 7-й бит произвольного целого числа в 1. Проверить работоспособность программы.

Математическая модель решения: аналогично предыдущему заданию, за тем исключением, что мы не отрицаем маску и производим операцию побитовой конъюнкции, а не дизъюнкции.

Листинг 1.2 – Код программы

```
void task1b() {  
    cout << "\n\nTask 1.б\n";  
    unsigned char x = 0;  
    unsigned char mask = 1;  
    x = x | (mask << 6);  
    cout << bitset<8>(x);  
}
```

Тестирование: в качестве входного значения возьмём 0 и посмотрим, что выведет программа (Рис. 3).



```
Task 1.б  
01000000  
E:\Education\
```

Рисунок 3 – Тестирование программы

Вывод: Тестирование показало, что данный подход корректен к данной задаче.

2.3 Задание 1.в

Формулировка задачи: реализовать приведённый в задании пример.

Математическая модель решения: с помощью маски и битового сдвига в ней, выводи двоичного представления целого десятичного числа.

Листинг 1.3 – Код программы

```
void task1c() {  
    cout << "\n\nTask 1.в\n";  
    unsigned int a = 25;  
    const int n = sizeof(int) * 8;  
    unsigned maska = (1 << n - 1); // Помещаем 1 в старший разряд  
    cout << "Start mask: " << bitset<n>(maska) << endl;  
    cout << "Result: ";  
    for (int i = 1; i <= n; i++) {  
        cout << ((a & maska) >> (n - i));  
        maska >>= 1;  
    }  
}
```

Тестирование: для тестирования возьмём число 25 (Рис. 4).

```
Task 1.B
Start mask: 10000000000000000000000000000000
Result: 000000000000000000000000000011001
E:\Education\СИАОД-2\ConsoleApplication1\x64\De
```

Рисунок 4 – Тестирование программы

В результате получаем 11001, что представляет из себя число 25 в двоичном представлении, следовательно программа работает корректно.

Вывод: для корректного вывода битовой сетки необходимо начинать с конца числа.

2.4 Задание 2.а

Формулировка задачи: отсортировать массив уникальных цифр от 0 до 7 в массиве размером не более 8-и элементов с помощью битового массива.

Математическая модель решения: для начала вводи размер массива. Для ввода элементов используем временную переменную, как пользователь вводит новую цифру, мы записываем её в битовый массив путём присваивания единицы биту, советуя этой цифре. При выводе проверяем значение бита и если оно равно 1, то выводим его номер.

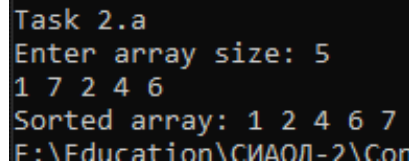
Листинг 1.4 – Код программы

```
void task2a() {
    cout << "\n\nTask 2.a\n";
    unsigned char data = 0;
    short int size, tmp;
    cout << "Enter array size: ";
    cin >> size;

    // Input array
    for (int i = 0; i < size; i++)
    {
        cin >> tmp;
        data = data | (1 << tmp);
    }

    // Outing sorted array
    cout << "Sorted array: ";
    for (int i = 0; i < sizeof(data) * 8; i++)
        if ((1 << i) & data) cout << i << " ";
}
```

Тестирование: для тестирования возьмём произвольный массив из 5-и элементов (Рис. 5).



```
Task 2.a
Enter array size: 5
1 7 2 4 6
Sorted array: 1 2 4 6 7
E:\Education\СИАОД-2\ConsoleApplication1\
```

Рисунок 5 – Тестирование программы

Вывод: для сортировка небольших массив уникальных чисел довольно эффективно использовать данный подход.

2.5 Задание 2.6

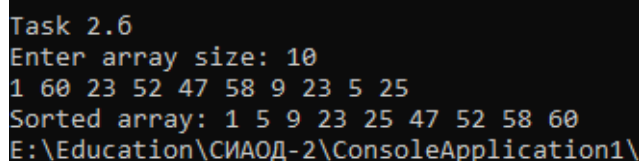
Формулировка задачи: отсортировать массив уникальных цифр от 0 до 63 в массиве размером не более 64-и элементов с помощью битового массива.

Математическая модель решения: аналогично предыдущему номеру, за исключением смены типа на unsigned long long (больше битовая сетка) и вместе обычной единицы, которую мы смещаем, используем маску советующего типа.

Листинг 1.5 – Код программы

```
void task2b() {
    cout << "\n\nTask 2.6\n";
    unsigned long long data = 0, mask = 1;
    short int size, tmp;
    cout << "Enter array size: ";
    cin >> size;
    for (int i = 0; i < size; i++)
    {
        mask = 1;
        cin >> tmp;
        data = data | (mask << tmp);
    }
    cout << "Sorted array: ";
    for (int i = 0; i < sizeof(data) * 8; i++)
    {
        mask = 1;
        if ((mask << i) & data) cout << i << " ";
    }
}
```

Тестирование: для тестирования возьмём произвольный массив из 10-и элементов (Рис. 6).



```
Task 2.6
Enter array size: 10
1 60 23 52 47 58 9 23 5 25
Sorted array: 1 5 9 23 25 47 52 58 60
E:\Education\СИАОД-2\ConsoleApplication1\
```

Рисунок 6 – Тестирование программы

Вывод: для сортировка средних массив уникальных чисел довольно эффективно использовать данный подход.

2.6 Задание 2.в

Формулировка задачи: Исправьте программу задания 2.б, чтобы для сортировки набора из 64-х чисел использовалось не одно число типа unsigned long, а линейный массив чисел типа unsigned char.

Математическая модель решения: для начала зарезервируем место для чисел, а далее с помощью побитового смещения и обращению по индексу, задаём единичные значения нужным битам.

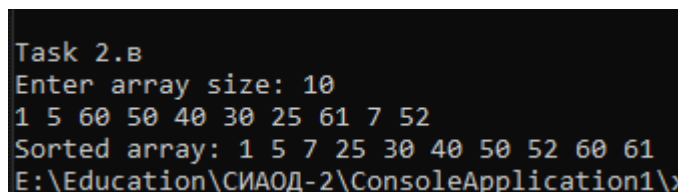
Листинг 1.6 – Код программы

```
void task2c() {
    cout << "\n\nTask 2.в\n";
    int size, tmp;
    vector<unsigned char> data;
    cout << "Enter array size: ";
    cin >> size;
    for (int i = 0; i < 8; i++)
        data.push_back(0);

    for (int i = 0; i < size; i++) {
        cin >> tmp;
        data[tmp / 8] = data[tmp / 8] | (1 << (tmp % 8));
    }

    cout << "Sorted array: ";
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            if ((1 << j) & data[i]) cout << i * 8 + j << " ";
}
```

Тестирование: для тестирования возьмём произвольный массив из 10-и элементов (Рис. 7).



```
Task 2.в
Enter array size: 10
1 5 60 50 40 30 25 61 7 52
Sorted array: 1 5 7 25 30 40 50 52 60 61
E:\Education\СИАОД-2\ConsoleApplication1\
```

Рисунок 7 – Тестирование программы

2.7 Задание 3.а и 3.б

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

Математическая модель решения: для начала генерируется случайную последовательность из уникальных чисел, для этого используем множество и генератор случайных чисел, затем сохраняем все числа в файл через строчку. Запускаем таймер и открываем 2 файла, для входных и выходных данных. А далее по аналогии с предыдущим номер повторяем логику обработки массива, за тем исключением, что числа считываются не из потока, а из файла.

Листинг 1.7 – Код программы

```
#include <set>
#include <iostream>
#include <bitset>
#include <vector>
#include <ctime>
#include <fstream>
#include <cmath>
#include <random>
#include <cstdlib>
#include <stdio.h>
#include <algorithm>
#include <chrono>
#include <windows.h>

#define RAND_MAX 10000000 //2147483647

using namespace std;
using namespace std::chrono;

bool file_generation(int size) {
    if (size >= 10000000) return false;

    // Generator
    set<int> numbers_set;
    default_random_engine u{};
    uniform_int_distribution<> d{};
    u.seed(random_device() ()); // Analog srand
    while (numbers_set.size() < size)
        numbers_set.insert(d(u, uniform_int_distribution<>::param_type{ 0,
RAND_MAX }));

    // Set to vector
    vector<int> numbers(numbers_set.begin(), numbers_set.end());
    shuffle(numbers.begin(), numbers.end(), u);

    // Create file
    ofstream out("input.txt");
    for (int num : numbers)
        out << num << endl;
    out.close();
    return true;
}

int main() {

    int size;

    cout << "Enter number of digits: ";
    cin >> size;
    if (!file_generation(size)) return 1;
```

Продолжение Листинга 1.7

```
// Start timer
auto start = high_resolution_clock::now();

// Open files
ifstream file_in("input.txt");
ofstream file_out("output.txt");

int tmp;
vector<unsigned char> data;

// Unpack numbers
while (file_in >> tmp) {
    if (tmp > data.size() * 8) data.resize(tmp / 8 + 1); // Resize
vector
    data[tmp / 8] = data[tmp / 8] | (1 << (tmp % 8)); // Set Bit
}

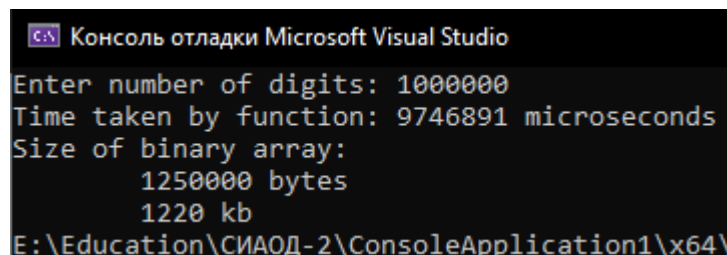
// Pack numbers
for (int i = 0; i < data.size(); i++)
    for (int j = 0; j < 8; j++)
        if ((1 << j) & data[i]) file_out << i * 8 + j << endl;

// Stop timer and count duration
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);

// Working time
cout << "Time taken by function: " << duration.count() << "
microseconds" << endl;

    cout << "Size of binary array: \n\t" << data.size() << " bytes\n\t" <<
data.size() / 1024 << " kb";
}
```

Тестирование: для теста зададим количество элементов 1 000 000, программа выполнилась за 9.74 секунд, а битовый массив занял 1220 килобайт (Рис. 8).



```
Консоль отладки Microsoft Visual Studio
Enter number of digits: 1000000
Time taken by function: 9746891 microseconds
Size of binary array:
    1250000 bytes
    1220 kb
E:\Education\СИАОД-2\ConsoleApplication1\x64\
```

Рисунок 8 – Тестирование программы

Вывод: если возникает потребность отсортировать файл с уникальными числами, то сортировка с помощью битового массива является довольно эффективной.

ВЫВОДЫ

При выполнении работы был получен опыт работы с битовыми операторами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием С++. 2-е изд., 2016.
2. Документация по языку С++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 01.09.2021).
3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: <https://online-edu.mirea.ru/course/view.php?id=4020> (дата обращения 01.09.2021).