



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 5.1

Тема: «Работа с данными из файла»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Фамилия И.О.

Группа AAAA-00-00

Москва 2024

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ.....	3
1.1 Цель работы.....	3
1.2 Постановка задания.....	3
2 ХОД РАБОТЫ.....	5
2.1 Задание 1.а.....	5
2.2 Задание 1.б.....	5
2.3 Задание 1.в.....	6
2.4 Задание 2.а.....	6
2.5 Задание 2.б.....	7
2.6 Задание 2.в.....	8
2.7 Задание 3.а.....	9
3 ВЫВОД.....	10
4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ.....	11

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Цель работы

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

1.2 Постановка задания

1.а. Реализуйте из пример листинга 1, проверьте правильность результата в том числе и на других значениях x .

Листинг 1 - Пример 1

```
unsigned char x=255;      //8-разрядное двоичное число 11111111
unsigned char maska = 1; //1=00000001 - 8-разрядная маска
x = x & (~ (maska<<4)); //результат x=239
```

1.б. Реализуйте по аналогии с предыдущим примером установку 7-го бита числа в единицу.

1.в. Реализуйте код листинга 2, объясните выводимый программой результат.

Листинг 2 - Пример 2

```
#include <iostream>
#include <bitset>

using namespace std;

int main()
{
    unsigned int x = 25;
    const int n = sizeof(int) * 8;
    unsigned int maska = (1 << n - 1);
    cout << "Начальный вид маски: " << bitset<n>(maska) <<
endl;
    cout << "Результат: ";
    for (int i = 1; i <= n; i++)
    {
        cout << ((x & maska) >> (n - i));
        maska = maska >> 1;
    }
    cout << endl;

    return 0;
}
```

2.а. Реализуйте 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.

2.б. Адаптируйте пример для набора из 64-х чисел (со значениями от 0 до 63) с битовым массивом в виде числа типа unsigned long long.

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

3.а. 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.

3.б. Определите программно объём оперативной памяти, занимаемый битовым массивом.

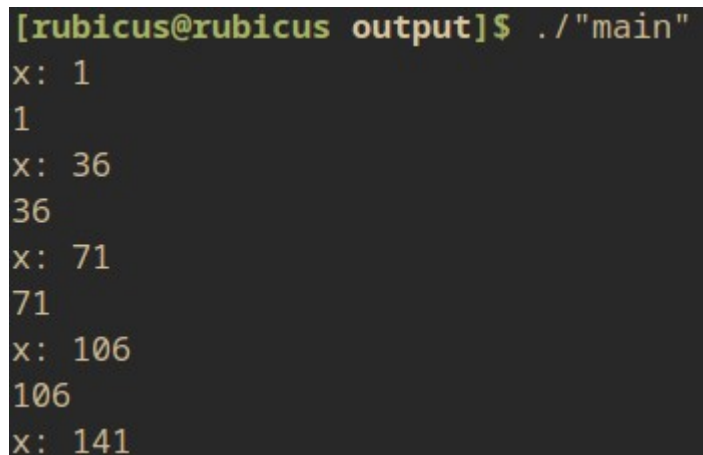
2 ХОД РАБОТЫ

2.1 Задание 1.а

Реализуем пример и проверим его правильность на других значениях x . Код программы приведён в листинге 3, результаты тестирования на рисунке 1.

Листинг 3 - Код программы 1.а

```
int main() {
    unsigned char x = 255;
    unsigned char maska = 1;
    x = x & ( ~ (maska << 4));
    cout << "x: " << (int)x << endl;
    cout << (int)x << endl;
    return 0;
}
```



```
[rubicus@rubicus output]$ ./"main"
x: 1
1
x: 36
36
x: 71
71
x: 106
106
x: 141
```

Рисунок 1 - Результаты тестирования задания 1.а

2.2 Задание 1.б

Реализуем по аналогии данный пример с установкой 7-го бита числа в единицу. Код программы приведён в листинге 4, результаты тестирования на рисунке 2.

Листинг 4 - Код программы 1.б

```
int main() {
    unsigned char x = 0;
    unsigned char maska = 1;
    x = x | (maska << 6);
    cout << "x: " << (int)x << endl;
    cout << (int)x << endl;
    return 0;
}
```

```
[rubicus@rubicus output]$ ./"main"
x: 0
64
x: 37
101
x: 74
74
x: 111
111
x: 148
212
```

Рисунок 2 - Результаты тестирования задания 1.б

2.3 Задание 1.в

Реализуем пример из листинга 2. После анализа кода и выводимого результата становится понятно как работает алгоритм.

Алгоритм выводит каждый бит числа используя маску, сдвигая единицу маски вправо каждую итерацию.

Результат при запуске равен 11001, что в переводе в десятичную систему равно 25, что и является введенным числом.

2.4 Задание 2.а

Реализуем пример с вводом до 8-ми их сортировки при помощи битового массива. Для корректной работы программы в начале также вводится количество вводимых элементов массива для сортировки. Код программы представлен в листинге 5. Результаты тестирования представлены на рисунке 3.

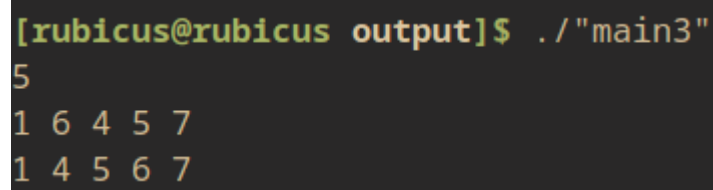
Листинг 5 - Код программы 2.а

```
int main()
{
    int n;
    unsigned char x = 0;
    unsigned int mask = 1;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int t;
        cin >> t;
```

```

        int m = mask << t;
        x = x | m;
    }
    for (int i = 8 - 1; i >= 0; i--)
    {
        if ((int)(x & mask) != 0)
        {
            cout << 8 - i - 1 << " ";
        }
        mask = mask << 1;
    }
    cout << endl;
    return 0;
}

```



```

[rubicus@rubicus output]$ ./"main3"
5
1 6 4 5 7
1 4 5 6 7

```

Рисунок 3 - Результаты тестирования задания 2.а

2.5 Задание 2.б

Адаптируем задание 2.а для работы с 64-ми числами. Код программы представлен в листинге 6, результаты тестирования на рисунке 4.

Листинг 6 - Код программы 2.б

```

int main()
{
    int n;
    unsigned long long int x = 0;
    unsigned long long int mask = 1;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int t;
        cin >> t;
        unsigned long long int m = mask << t;
        x = x | m;
    }
    for (int i = 64 - 1; i >= 0; i--)
    {
        if (x & mask)
            cout << 64 - i - 1 << " ";
        mask = mask << 1;
    }
    return 0;
}

```

```
[rubicus@rubicus output]$ ./"main4"  
7  
1 15 63 52 42 41 37  
1 15 37 41 42 52 63
```

Рисунок 4 - Результаты тестирования задания 2.б

2.6 Задание 2.в

Исправим алгоритм задания 2.б. Для решения используем массив элементов типа `unsigned char` произвольной длины. В реализации была взята длина 128. Таким образом мы сможем хранить $8 \cdot 128$ элементов размером $8 \cdot 128$. В общем случае $8 \cdot n$, где размер n — ограничен лишь размером доступной памяти.

Код программы представлен в листинге 7. Результаты тестирования представлены на рисунке 5.

Листинг 7 - Код программы 2.в

```
int main()  
{  
    int n;  
    unsigned char xarr[128]{0};  
    unsigned char mask = 1;  
    cin >> n;  
    for (int i = 0; i < n; i++)  
    {  
        int t;  
        cin >> t;  
        int j = floor(t / 8);  
        int k = t - 8 * j;  
        int m = mask << (8 - k - 1);  
        xarr[j] = xarr[j] | m;  
    }  
    for (int i = 0; i < 128; i++)  
    {  
        mask = 128;  
        for (int j = 0; j < 8; j++)  
        {  
            if (xarr[i] & mask)  
                cout << 8 * i + j << " ";  
            mask = mask >> 1;  
        }  
    }  
}
```



```
cout << endl;  
return 0;  
}
```

```
[rubicus@rubicus output]$ ./"main5"  
16  
182 129 391 241 24 12 5 124 813 158 293 245 787 312 555 333  
5 12 24 124 129 158 182 241 245 293 312 333 391 555 787 813
```

Рисунок 5 - Результаты тестирования задания 2.в

2.7 Задание 3.а

Для генерации входного файла используем возможности командой строки: `echo {1..8000000} | xargs -n 1 | shuf > nums.txt`. Получим 8 миллионов не повторяющихся перемешанных чисел.

Исправим код из задания 2.в для работы с файлом вместо ручного ввода. Чтобы не тратить лишние ресурсы на чтение файла, сортировку будем выполнять прямо при чтении очередной строки.

Вывод также организуем на запись в файл вместо консоли.

Код программы представлен в листинге 8, результаты тестирования на рисунке 5 и 6. На рисунке 5 — файл до сортировки, на рисунке 6 — после.

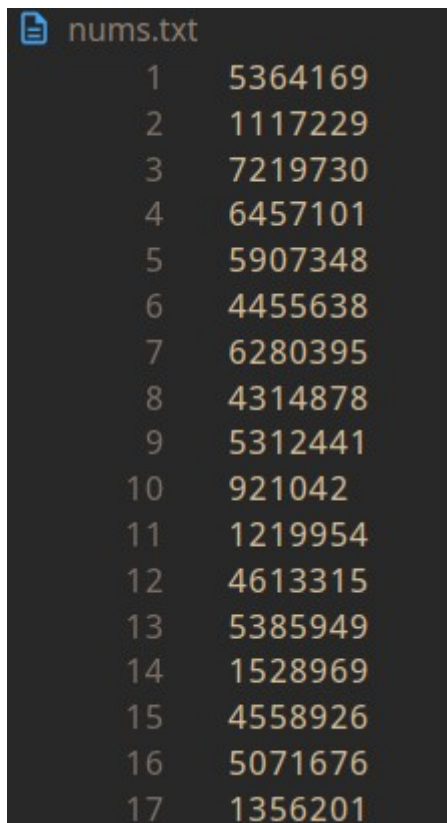
Листинг 8 - Код программы 3.а

```
int main()  
{  
    const int size = 1024 * 1024;  
    vector<unsigned char> xarr(size);  
    unsigned char mask = 1;  
    int t, j, k, m;  
  
    ifstream infile("nums.txt");  
    string line;  
    while(getline(infile, line))  
    {  
        unsigned int t = stoi(line);  
        j = floor(t / 8);  
        k = t - 8 * j;  
        m = mask << (8 - k - 1);  
        xarr[j] = xarr[j] | m;  
    }  
  
    infile.close();  
    ofstream output("sorted.txt");
```

```

for (unsigned int i = 0; i < size; i++)
{
    mask = 128;
    for (unsigned short j = 0; j < 8; j++)
    {
        if (xarr[i] & mask)
            output << 8 * i + j << "\n";
        mask = mask >> 1;
    }
}
return 0;
}

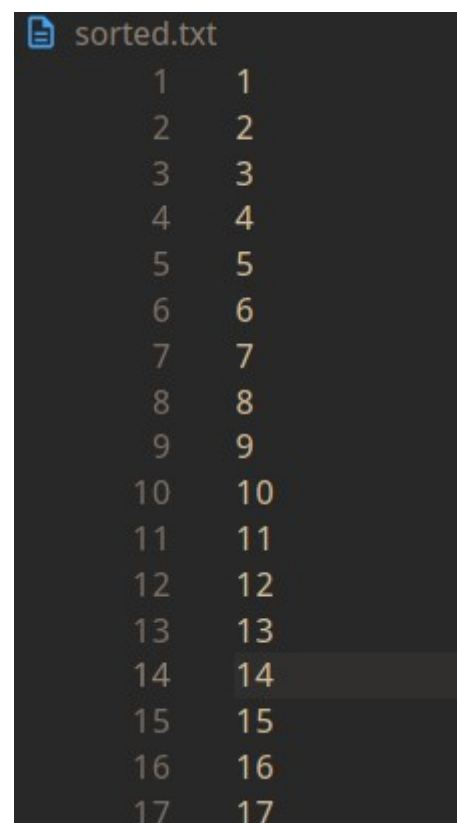
```



nums.txt

1	5364169
2	1117229
3	7219730
4	6457101
5	5907348
6	4455638
7	6280395
8	4314878
9	5312441
10	921042
11	1219954
12	4613315
13	5385949
14	1528969
15	4558926
16	5071676
17	1356201

Рисунок 6 - Файл до сортировки



sorted.txt

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17

Рисунок 7 - Файл после сортировки

2.8 Задание 3.6

Дополним код программы 3.а для вычисления времени работы. Сделаем это при помощи библиотеки `chrono` и будем вычислять разницу между временем начала работы программы и концом работы программы.

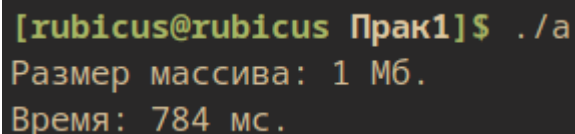
Для вычисления размера занятого массива воспользуемся встроенной в `c++` функцией `sizeof`, и приведём полученное значение к Мб. Код программы приведён в листинге 9. Результаты тестирования на рисунке 8.

Листинг 9 - Код программы 3.б

```
int main()
{
    auto start = high_resolution_clock::now();
    const int size = 1024 * 1024;
    vector<unsigned char> xarr(size);
    unsigned char mask = 1;
    int t, j, k, m;

    ifstream infile("nums.txt");
    string line;
    while (getline(infile, line))
    {
        unsigned int t = stoi(line);
        j = floor(t / 8);
        k = t - 8 * j;
        m = mask << (8 - k - 1);
        xarr[j] = xarr[j] | m;
    }

    infile.close();
    ofstream output("sorted.txt");
    for (unsigned int i = 0; i < size; i++)
    {
        mask = 128;
        for (unsigned short j = 0; j < 8; j++)
        {
            if (xarr[i] & mask)
                output << 8 * i + j << "\n";
            mask = mask >> 1;
        }
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop -
start).count();
    cout << "Размер массива: " << (sizeof(unsigned char)) *
xarr.size() / size << " Мб." << endl;
    cout << "Время: " << duration << " мс." << endl;
    return 0;
}
```



```
[rubicus@rubicus Прак1]$ ./a
Размер массива: 1 Мб.
Время: 784 мс.
```

Рисунок 8 - Результаты
тестирования

3 ВЫВОД

Были освоены приёмы работы с битовым представлением беззнаковых целых чисел, а также был реализован эффективный алгоритм внешней сортировки на основе битового массива.

4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Структуры и алгоритмы обработки данных (часть 2): Лекционные материалы / Рысин М. Л. МИРЭА — Российский технологический университет, 2022/23. – 82 с.