



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение  
высшего образования*

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

---

---

Отчет по выполнению практического задания №5

**Тема:**

**РАБОТА С ДАННЫМИ ИЗ ФАЙЛА**

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Враженко Д.О.

Группа: ИКБО-50-23

Вариант: 6

Москва – 2024

## ЦЕЛЬ РАБОТЫ

**Часть 5.1.** Освоить приёмы работы с битовым представлением беззнаковых целых чисел, реализовать эффективный алгоритм внешней сортировки на основе битового массива.

**Часть 5.2.** Получить практический опыт по применению алгоритмов поиска в таблицах данных.

## ХОД РАБОТЫ

### Часть 5.1. Битовые операции. Сортировка числового файла с помощью битового массива

#### Задание 1:

##### Поставка задачи:

**1.а.** Реализуйте вышеприведённый пример, проверьте правильность результата в том числе и на других значениях  $x$ .

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

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

##### Математическая модель решения:

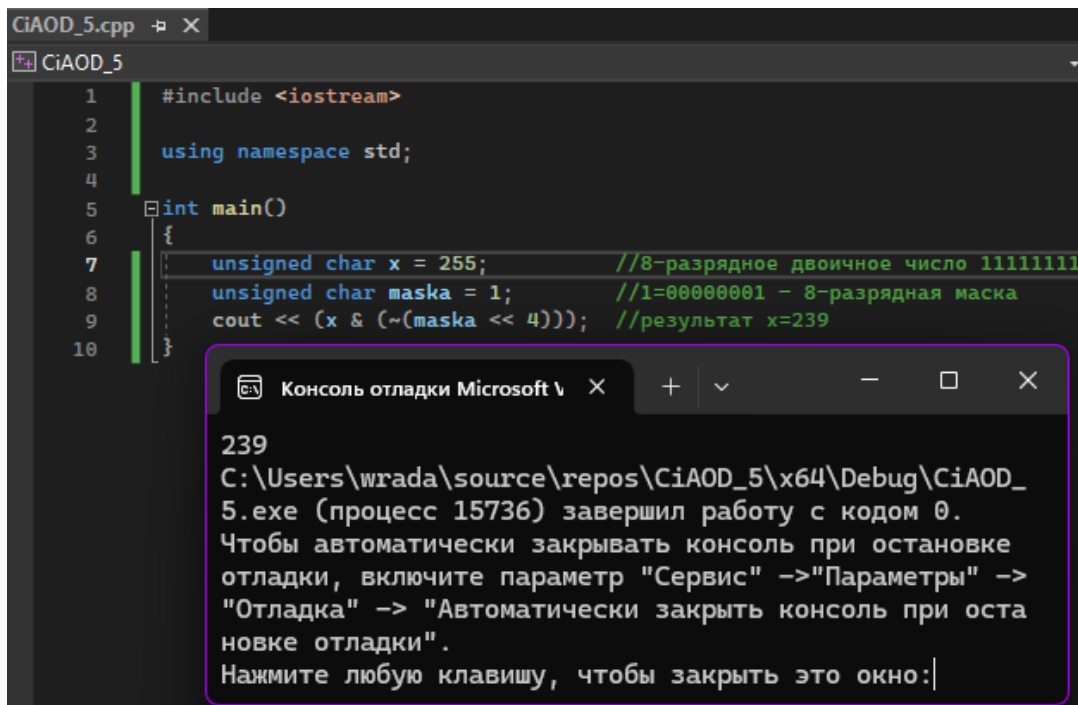
**1.а.** Чтобы установить 5-й бит числа в 0, нужно воспользоваться побитовой операцией «и» с маской, в которой все биты равны 1, кроме 5-го, который равен 0. Например, маска для 5-го бита выглядит как  $\sim(1 \ll 4)$ .

**1.б.** Чтобы установить 7-й бит числа в 1, нужно использовать побитовую операцию «или» с маской, где только 7-й бит равен 1. Маска для 7-го бита — это  $1 \ll 6$ .

**1.в.** Программа создает маску, в которой только самый старший бит равен 1, а остальные 0. Затем она выводит это число в виде битовой строки. В цикле проверяется каждый бит числа, начиная с самого старшего, путем побитовой «и» с текущей маской, после чего маска сдвигается вправо.

##### Код программы:

**1.а.** На рис. 1 приведён пример переписанного кода из файла с заданием с небольшим изменением для вывода результата на экран.



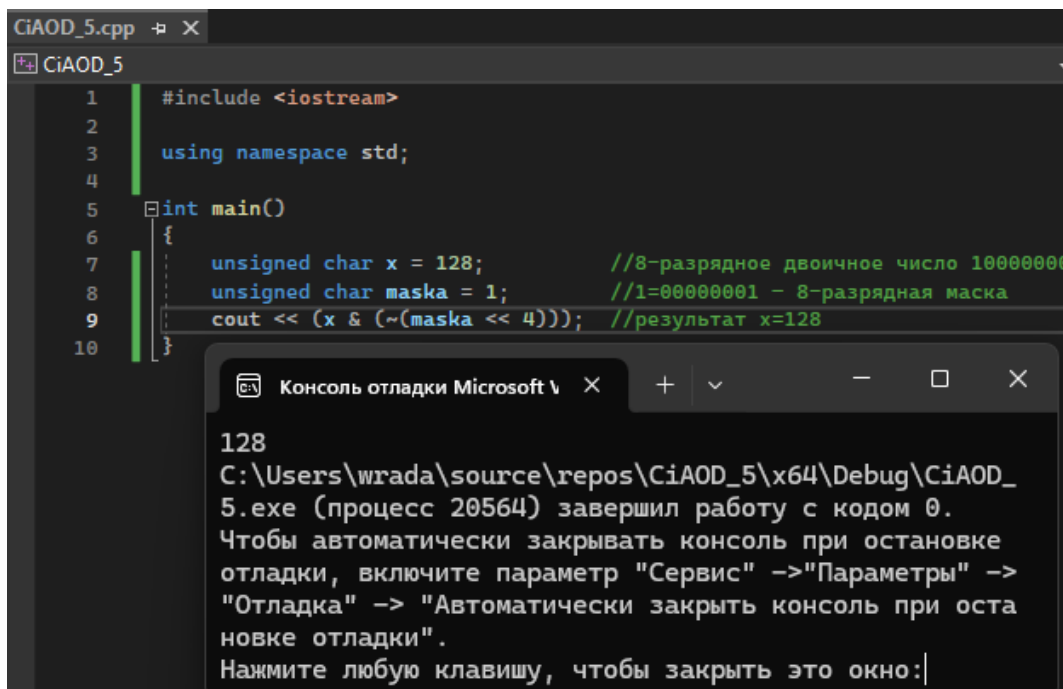
```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      unsigned char x = 255;           //8-разрядное двоичное число 11111111
8      unsigned char maska = 1;         //1=00000001 - 8-разрядная маска
9      cout << (x & (~(maska << 4))); //результат x=239
10 }
```

Консоль отладки Microsoft V

239  
C:\Users\wrada\source\repos\CiAOD\_5\x64\Debug\CiAOD\_5.exe (процесс 15736) завершил работу с кодом 0.  
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно:

Рисунок 1 - Код из файла с заданием и результат

Теперь проверим программу на других значениях x. Например,  $x = 128$ . На рис. 2 приведён результат выполнения программы с изменёнными входными данными.



```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      unsigned char x = 128;           //8-разрядное двоичное число 10000000
8      unsigned char maska = 1;         //1=00000001 - 8-разрядная маска
9      cout << (x & (~(maska << 4))); //результат x=128
10 }
```

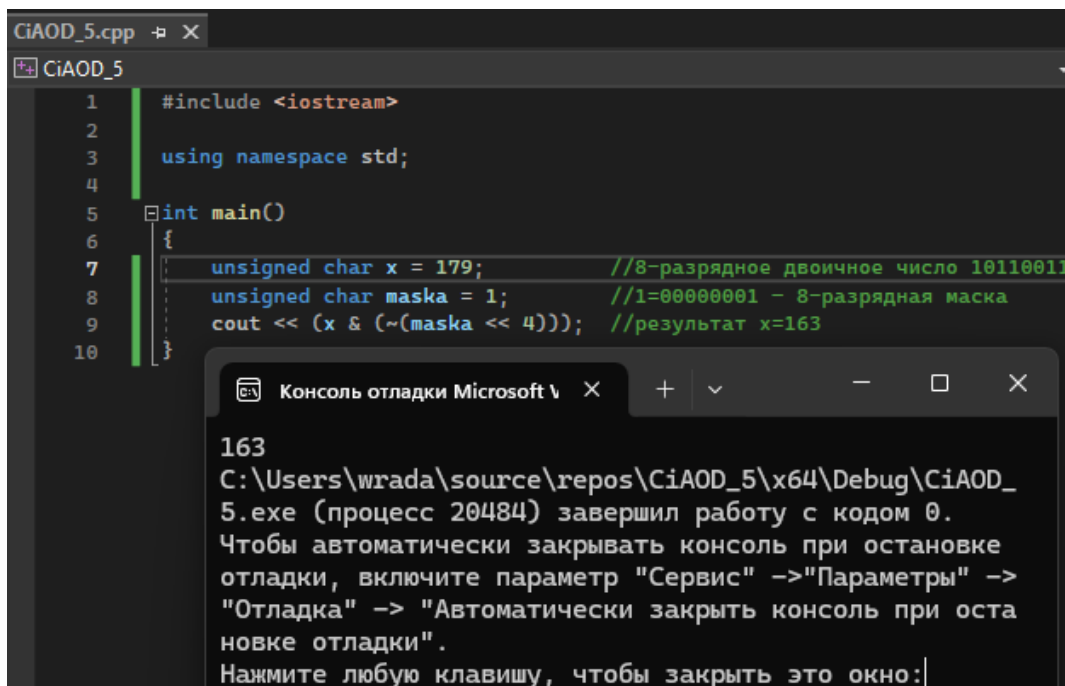
Консоль отладки Microsoft V

128  
C:\Users\wrada\source\repos\CiAOD\_5\x64\Debug\CiAOD\_5.exe (процесс 20564) завершил работу с кодом 0.  
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно:

Рисунок 2 - Результат работы программы при  $x=128$

Как видно результат не изменился, ведь в двоичном коде числе 128 на месте 5-го разряда и так стоит 0.

Приведём другой пример:  $x=179$ . На рис. 3 приведён результат выполнения программы с изменёнными входными данными.



```
CiAOD_5.cpp
CiAOD_5
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      unsigned char x = 179;           //8-разрядное двоичное число 10110011
8      unsigned char maska = 1;        //1=00000001 - 8-разрядная маска
9      cout << (x & ~(maska << 4));    //результат x=163
10 }
```

```
Консоль отладки Microsoft V
163
C:\Users\wrada\source\repos\CiAOD_5\x64\Debug\CiAOD_5.exe (процесс 20484) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 3 - Результат работы программы при  $x=179$

**1.6.** Для выполнения данного задания код программы был немного изменён. Теперь сдвиг влево происходит на 6 бит, вместо 4, мы убрали инверсию, а также заменили побитовое И на побитовое ИЛИ, то есть  $\&$  на  $|$ . На рис. 4 приведён результат работы программы с внесёнными изменениями при входном значении  $x=128$ .

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      unsigned char x = 128;           //8-разрядное двоичное число 10000000
8      unsigned char maska = 1;        //1=00000001 - 8-разрядная маска
9      cout << (x | (maska << 6));     //результат x=192
10 }

```

```

192
C:\Users\wrada\source\repos\CiAOD_5\x64\Debug\CiAOD_5.exe (процесс 4556) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:

```

Рисунок 4 - Результат работы программы при  $x=128$

1.в. На рис. 5 представлен реализованный код листинга 1 из файла практической работы.

```

int ciaod_1v()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    unsigned int x = 25;
    const int n = sizeof(int) * 8; //32 - количество разрядов в числе типа int
    unsigned maska = (1 << n - 1); //1 в старшем бите 32-разрядной сетки
    cout << "Начальный вид маски: " << bitset<n>(maska) << endl;
    cout << "Результат: ";
    for (int i = 1; i <= n; i++) //32 раза - по количеству разрядов:
    {
        cout << ((x & maska) >> (n - i));
        maska = maska >> 1; //смещение 1 в маске на разряд вправо
    }
    cout << endl;
    system("pause");
    return 0;
}

```

```

Выбрать D:\Visual Studio\Programs\CiAOD_5\CiAOD_...
Начальный вид маски: 10000000000000000000000000000000
Результат: 000000000000000000000000000011001
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5 - Результат работы кода из листинга 1

В выводе программы мы видим, что сначала выводится начальный вид маски в виде 1 единицы и 31 нуля, ведь в самом коде можно заметить, что мы создаём 32-битную маску с 1 единицей и сразу же сдвигаем её 31 бит влево. Во второй строке вывода программы мы видим результат, в котором написано заданное пользователем число (в нашем случае 25) в двоичном коде в 32-битном

выводе. Это значит, что если вводимое число меньше, чем  $2^{32}$ , то оно поместится в нашу маску, состоящую из 32 бит, и в случае необходимости дополнительно заполнится незначащими нулями, а если число будет больше, чем  $2^{32}$ , то в выводе мы будем видеть только крайние справа 32 цифры.

## **Задание 2:**

### **Постановка задачи:**

**2.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.

**2.б.** Адаптируйте вышеприведённый 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.

### **Математическая модель решения:**

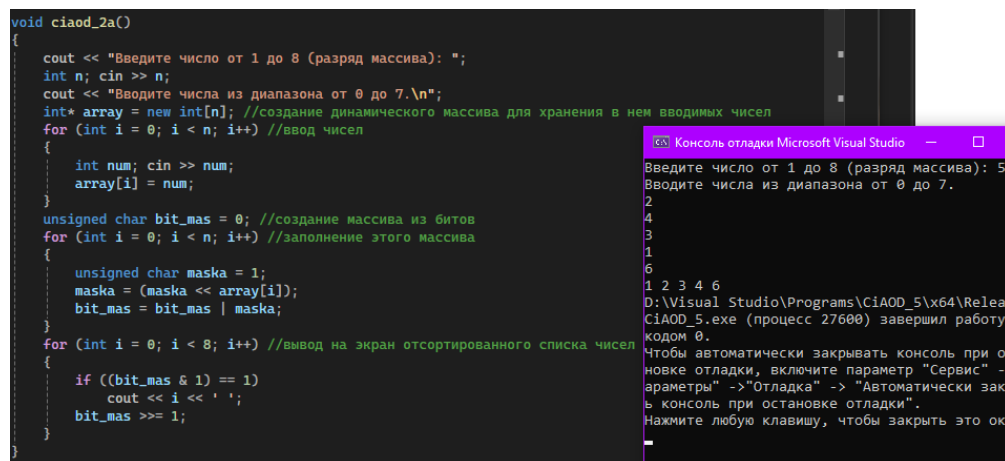
**2.а.** Для сортировки до 8 чисел, каждое из которых находится в диапазоне от 0 до 7, создается переменная типа unsigned char, где каждый бит представляет одно число. Инициализируется переменная unsigned char равная 0. Для каждого числа в наборе устанавливается соответствующий бит в 1 с помощью побитовой операции «или». Затем программа последовательно проходит по битам переменной и выводит индексы единичных битов, что дает отсортированный набор чисел.

**2.б.** Для работы с числовым диапазоном от 0 до 63 используется тип unsigned long long, где каждый бит отвечает за одно число. Инициализируется переменная unsigned long long равная 0. Для каждого числа в наборе устанавливается соответствующий бит в 1. После этого программа проходит по всем 64 битам и выводит индексы единичных битов, что обеспечивает сортировку.

**2.в.** Вместо одного большого числа типа unsigned long long используется массив из 8 переменных типа unsigned char, каждая из которых отвечает за 8 бит. Инициализируется массив из 8 элементов unsigned char. Для каждого числа вычисляется индекс элемента массива и позиция бита в этом элементе. После установки соответствующего бита программа выводит индексы единичных битов, что позволяет получить отсортированный набор чисел.

### Код программы:

**2.а.** На рис. 6 представлен код программы для решения данной задачи и пример тестирования.



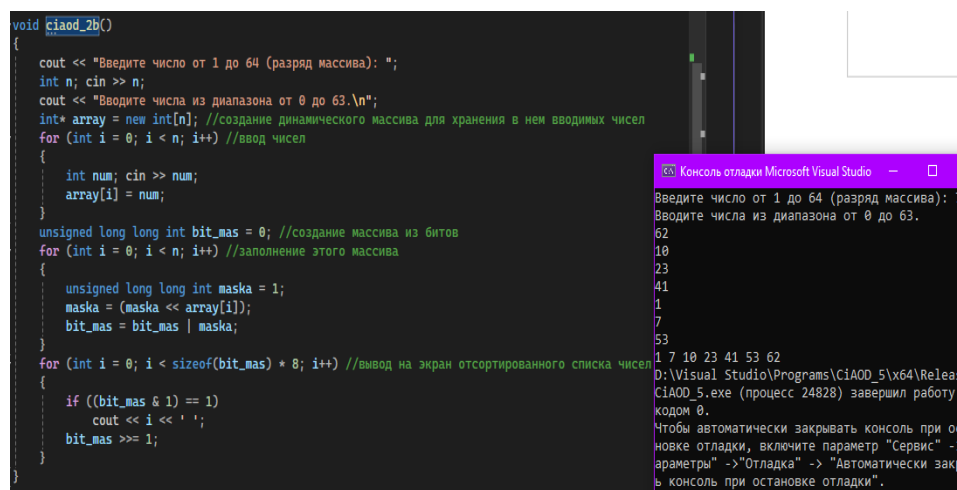
```
void ciaod_2a()
{
    cout << "Введите число от 1 до 8 (разряд массива): ";
    int n; cin >> n;
    cout << "Введите числа из диапазона от 0 до 7.\n";
    int* array = new int[n]; //создание динамического массива для хранения в нем вводимых чисел
    for (int i = 0; i < n; i++) //ввод чисел
    {
        int num; cin >> num;
        array[i] = num;
    }
    unsigned char bit_mas = 0; //создание массива из битов
    for (int i = 0; i < n; i++) //заполнение этого массива
    {
        unsigned char maska = 1;
        maska = (maska << array[i]);
        bit_mas = bit_mas | maska;
    }
    for (int i = 0; i < 8; i++) //вывод на экран отсортированного списка чисел
    {
        if ((bit_mas & 1) == 1)
            cout << i << ' ';
        bit_mas >>= 1;
    }
}
```

Консоль отладки Microsoft Visual Studio

Введите число от 1 до 8 (разряд массива): 5  
Введите числа из диапазона от 0 до 7.  
2  
4  
3  
1  
6  
1 2 3 4 6  
D:\Visual Studio\Programs\CiA0D\_5\Release  
CiA0D\_5.exe (процесс 27600) завершил работу  
кодом 0.  
Чтобы автоматически закрывать консоль при ос  
новке отладки, включите параметр "Сервис" ->  
араметры" ->"Отладка" -> "Автоматически закр  
ь консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окн

Рисунок 6 - Код для задачи 2.а и пример тестирования

**2.б.** На рис. 7 представлен код программы для решения данной задачи и пример тестирования.



```
void ciaod_2b()
{
    cout << "Введите число от 1 до 64 (разряд массива): ";
    int n; cin >> n;
    cout << "Введите числа из диапазона от 0 до 63.\n";
    int* array = new int[n]; //создание динамического массива для хранения в нем вводимых чисел
    for (int i = 0; i < n; i++) //ввод чисел
    {
        int num; cin >> num;
        array[i] = num;
    }
    unsigned long long int bit_mas = 0; //создание массива из битов
    for (int i = 0; i < n; i++) //заполнение этого массива
    {
        unsigned long long int maska = 1;
        maska = (maska << array[i]);
        bit_mas = bit_mas | maska;
    }
    for (int i = 0; i < sizeof(bit_mas) * 8; i++) //вывод на экран отсортированного списка чисел
    {
        if ((bit_mas & 1) == 1)
            cout << i << ' ';
        bit_mas >>= 1;
    }
}
```

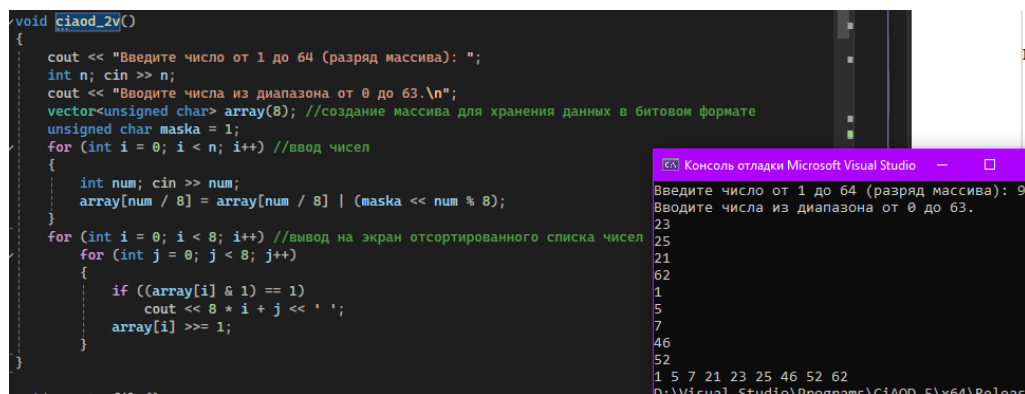
Консоль отладки Microsoft Visual Studio

Введите число от 1 до 64 (разряд массива): 7  
Введите числа из диапазона от 0 до 63.  
62  
10  
23  
41  
1  
7  
53  
1 7 10 23 41 53 62  
D:\Visual Studio\Programs\CiA0D\_5\Release  
CiA0D\_5.exe (процесс 24828) завершил работу  
кодом 0.  
Чтобы автоматически закрывать консоль при ос  
новке отладки, включите параметр "Сервис" ->  
араметры" ->"Отладка" -> "Автоматически закр  
ь консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окн

Рисунок 7 - Код для задачи 2.б и пример тестирования



2.в. На рис. 8 представлен код программы для решения данной задачи и пример тестирования.



```
void ciaod_2v()
{
    cout << "Введите число от 1 до 64 (разряд массива): ";
    int n; cin >> n;
    cout << "Введите числа из диапазона от 0 до 63.\n";
    vector<unsigned char> array(8); //создание массива для хранения данных в битовом формате
    unsigned char maska = 1;
    for (int i = 0; i < n; i++) //ввод чисел
    {
        int num; cin >> num;
        array[num / 8] = array[num / 8] | (maska << num % 8);
    }
    for (int i = 0; i < 8; i++) //вывод на экран отсортированного списка чисел
    {
        for (int j = 0; j < 8; j++)
        {
            if ((array[i] & 1) == 1)
                cout << 8 * i + j << ' ';
            array[i] >>= 1;
        }
    }
}
```

Консоль отладки Microsoft Visual Studio

Введите число от 1 до 64 (разряд массива): 9  
Введите числа из диапазона от 0 до 63.  
23  
25  
21  
62  
1  
5  
7  
46  
52  
1 5 7 21 23 25 46 52 62  
D:\Visual\_Studio\Programs\CiAOD\_5\64\Release

Рисунок 8 - Код для задачи 2.в и пример тестирования

### Задание 3:

#### Поставка 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.

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

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

#### Математическая модель решения:

**3.а.** Для сортировки чисел в файле считываем данные и используем битовый массив, где каждый бит соответствует наличию числа в наборе. Вводится битовый массив размером 1 МБ (8 миллионов бит), который позволяет хранить числа до 8 миллионов. Числа из входного файла отмечаются установкой соответствующего бита в 1. После этого программа проходит по битовому массиву и записывает числа в отсортированном порядке в выходной файл. Время работы программы измеряется с помощью встроенного таймера.

**3.б.** Для определения объема оперативной памяти, занимаемой битовым массивом, вычисляем количество бит, необходимое для хранения чисел (1 бит на число). Поскольку массив ограничен 1 МБ, это позволяет хранить до 8 миллионов чисел ( $8 * 1024 * 1024 = 8388608$  битов). Программа выводит размер памяти, занимаемой массивом, в байтах.

### Код программы:

На рис. 9 представлен код для создания файла.

```
void create_file()
{
    const long n = 1000000; //минимальное значение чисел
    const long n_max = 9999999; //максимальное значение чисел
    const long length = n_max - n + 1; //максимальное количество чисел
    long* array = new long[length]; //динамический массив для хранения чисел
    for (long i = 0; i < length; i++) //заполнение массива числами
        array[i] = n + i;
    srand(time(NULL));
    for (long i = 0; i < length; i++) //перемешивание чисел в массиве
        swap(array[i], array[rand() % length]);
    ofstream file_out("test.txt");
    for (int i = 0; i < length; i++) //занесение чисел в файл
        file_out << array[i] << endl;
    file_out.close();
}
```

Рисунок 9 - Код для создания файла

На рис. 10 представлен код для решения задачи 3 и результат тестирования.

```

void ciaod_3()
{
    create_file();
    const int n = 10000000 / 8;
    int start = clock(); //запуск таймера
    unsigned char maska = 1;
    vector<unsigned char> bit_array(n); //создание битового массива размера n строк
    ifstream file_in("test.txt");
    int character;
    while (!file_in.eof()) //считывание данных из файла
    {
        file_in >> character;
        bit_array[character / 8] = bit_array[character / 8] | (maska << character % 8);
    }
    file_in.close();
    int stop = clock(); //остановка таймера
    ofstream file_out("test.txt");
    for (int i = 0; i < n; i++) //вывод чисел на экран
        for (int j = 0; j < 8; j++)
        {
            if ((bit_array[i] & 1) == 1)
                file_out << 8 * i + j << endl;
            bit_array[i] >>= 1;
        }
    file_out.close();
    int work_time = stop - start; //время работы
    bit_array.shrink_to_fit();
    cout << bit_array.capacity() << " b\n" << bit_array.capacity() / 1024 << " kb\n"
        << bit_array.capacity() / (1024 * 1024) << " mb\nTime: " << work_time << "ms";
}

```

Консоль отладки M

```

1250000 b
1220 kb
1 mb
Time: 4189ms
D:\Visual Studio\
exe (процесс 4636
Чтобы автоматичес
ладки, включите п
адка" -> "Автомат
отладки".
Нажмите любую кла

```

Рисунок 10 - Код для задачи 3 и результат тестирования

## Часть 5.2. Алгоритмы поиска в таблице при работе с данными из файла

Вариант 6: Бинарный однородный без использования дополнительной таблицы; Товар: название, код – шестиразрядное число.

### Задание 1:

#### Поставка задачи:

Создать двоичный файл из записей (структура записи определена вариантом). Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны.

Рекомендация: создайте сначала текстовый файл, а затем преобразуйте его в двоичный.

#### Подход к решению:

Для создания файлов был использован код для быстрой генерации случайных названий продуктов и их порядковых номеров для удобства распределения их в таблице, а также случайных неповторяющихся шестиразрядных чисел.

Затем была проведена сортировка данных в файле с использованием LibreOffice Calc.

### **Структура записи файла:**

«Название продукта»\_«Порядковый номер»,«Шестиразрядное число»

Такой формат записи применяется во всех требуемых файлах, то есть в файле на 100 записей, 1000 и 10000.

### **Размер записи в байтах:**

100 записей: 4 096 байт.

1000 записей: 20 480 байт.

10000 записей: 217 088 байт.

### **Скриншот файла:**

На рис. 11 представлена часть текстового файла с входными данными, определёнными вариантом.

```
Smartwatch_33,230053
Mouse_34,938534
Mouse_35,274766
Smartphone_36,153519
Headphones_37,434873
Tablet_38,757082
Printer_39,505231
Smartphone_40,239998
Mouse_41,240575
Printer_42,885456
Printer_43,478496
Smartwatch_44,207917
Printer_45,870177
Laptop_46,411211
```

Рисунок 11 - Часть текстового файла

### **Задание 2:**

#### **Поставка задачи:**

Поиск в файле с применением линейного поиска.

1. Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска.

2. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.

3. Составить таблицу с указанием результатов замера времени.

### Код функции поиска:

На рис. 12 представлен код функции поиска, определённый вариантом.

```
bool binary_homogeneous_search_without_using_an_additional_table(const string& binary_file, int total_lines, int search_code, Product& found_product)
{
    ifstream file(binary_file);
    int left = 0, right = total_lines - 1;
    while (left <= right)
    {
        int mid = left + (right - left) / 2; //определение центра относительно значений переменных left и right
        Product product;
        if (!read_product_from_file(file, mid, product)) //читаен продукт с позиции mid
            return false;
        if (product.code == search_code) //элемент найден
        {
            found_product = product;
            return true;
        }
        else if (product.code > search_code) //ищем в левой половине
            right = mid - 1;
        else //ищем в правой половине
            left = mid + 1;
    }
    return false; //продукт не найден
}
```

Рисунок 12 - Код функции поиска

### Сводная таблица с результатами тестирования:

Кол-во записей	Время работы, мс
100	1
1000	2
10000	87

### Задание 3:

#### Поставка задачи:

Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти.

1. Для оптимизации поиска в файле создать в оперативной памяти структуру данных – таблицу, содержащую ключ и ссылку (смещение) на запись в файле.

2. Разработать функцию, которая принимает на вход ключ и ищет в таблице элемент, содержащий ключ поиска, а возвращает ссылку на запись в файле. Алгоритм поиска определен в варианте.

3. Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. Возвращает прочитанную запись как результат.

4. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.

5. Составить таблицу с указанием результатов замера времени.

#### **Код функции для создания таблицы:**

На рис. 13 представлен код функции для создания таблицы.

```
vector<index_entry> build_index_table(string filename)
{
    ifstream file(filename);
    vector<index_entry> index_table;
    string line; streampos position;
    while (file) //проходим по файлу и сохраняем код продукта и его смещение
    {
        position = file.tellg(); //запоминаем позицию перед чтением строки
        if (!getline(file, line))
            break;
        size_t comma_pos = line.find(','); //нахождение координат запятой в строке
        if (comma_pos != string::npos)
        {
            int code = stoi(line.substr(comma_pos + 1));
            index_table.push_back({ code, position });
        }
    }
    file.close();
    return index_table;
}
```

Рисунок 13 - Код для создания таблицы

#### **Код функции для нахождения продукта в таблице**

На рис. 14 представлен код функции для нахождения продукта в таблице.

```
streampos find_in_table(const vector<index_entry>& index_table, int target_code)
{
    for (const auto& entry : index_table)
        if (entry.code == target_code)
            return entry.pos;
    return -1; //если код не найден
}
```

Рисунок 14 - Код для поиска в таблице

#### **Код функции для чтения продукта по позиции**

На рис. 15 представлен код функции для чтения информации о продукте.

```

Product read_product_by_position(string filename, streampos pos)
{
    ifstream file(filename);
    file.seekg(pos); //перемещение указателя в файле на позицию pos
    string line; getline(file, line);
    size_t comma_pos = line.find(','); //нахождение координат запятой в строке
    string name = line.substr(0, comma_pos);
    int code = stoi(line.substr(comma_pos + 1));
    file.close();
    return { name, code };
}

```

Рисунок 15 - Код для чтения информации о продукте

**Сводная таблица с результатами тестирования:**

Кол-во записей	Время работы, мс
100	0
1000	0
10000	0

## **ВЫВОД**

В работе рассмотрены методы побитовой обработки данных и их применение для сортировки числовых массивов. Реализованы эффективные алгоритмы сортировки, основанные на использовании битовых массивов, что позволило существенно оптимизировать потребление памяти. Применение битовых операций обеспечило высокую производительность при работе с большими объемами данных.

Кроме того, было показано, что алгоритмы поиска, использующие структуры данных в оперативной памяти, значительно превосходят по эффективности те, которые не задействуют дополнительные таблицы.



## СПИСОК ЛИТЕРАТУРЫ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
2. Документация по языку C++ [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).