



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

**Тема: «РАБОТА С ДАННЫМИ ИЗ ФАЙЛА»**

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент  
группа

Кузнецов Л. А.  
ИКБО-20-23

**Москва 2024**

## СОДЕРЖАНИЕ

Задание 1 части 5.1.....	2
Задание 1.а.....	2
Задание 1.б.....	3
Задание 1.в.....	4
Задание 2.....	4
Задание 2.а.....	6
Задание 2.б.....	7
Задание 2.в.....	9
Задание 3.....	11
Задание 3.а.....	11
Задание 3.б.....	11
Задание 1 части 5.2.....	13
Задание 1.1.....	13
Задание 1.2.....	13
Задание 1.3.....	15
ВЫВОДЫ.....	16
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ.....	16

## Задание 1 части 5.1

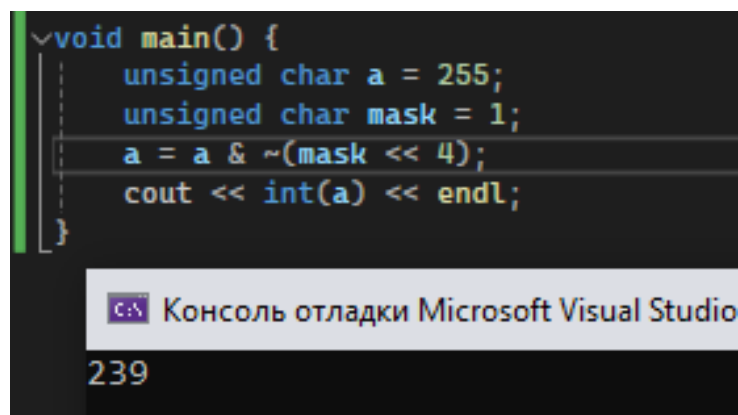
### Задание 1.а.

**Условие:** Реализуйте вышеприведённый пример (рис. 1), проверьте правильность результата в том числе и на других значениях  $x$ .

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

Рисунок 1 – Вышеприведённый пример

Как и требовалось, реализуем пример из рисунка 1 с выводом на экран получаемого значения.



```
void main() {
    unsigned char a = 255;
    unsigned char mask = 1;
    a = a & ~(mask << 4);
    cout << int(a) << endl;
}
```

Консоль отладки Microsoft Visual Studio

239

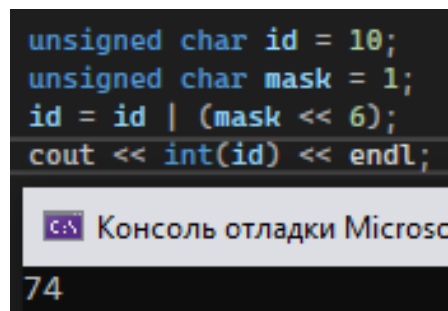
Рисунок 2 – Реализация кода из рисунка 1

Ответ соответствует ожиданиям, что означает – код был написан корректно.

### Задание 1.б.

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

Проделав небольшие изменения в коде, выполним требуемое. Результаты различных тестов приведены на рисунках 2-4.



```
unsigned char id = 10;
unsigned char mask = 1;
id = id | (mask << 6);
cout << int(id) << endl;
```

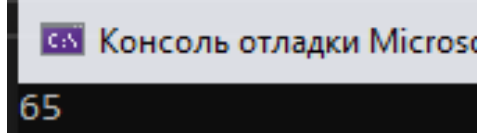
Консоль отладки Microsc

74

Рисунок 3 – Установка 7-го бита единицей id=10

В следующем примере на рис.4 id примет 64, чтобы увидеть, как число, уже имеющее 7-ой бит равный единице, не изменится.

```
unsigned char id = 65;
unsigned char mask = 1;
id = id | (mask << 6);
cout << int(id) << endl;
```



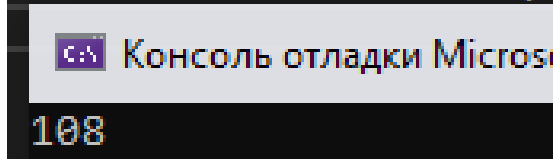
Консоль отладки Microsoft Visual Studio

65

**Рисунок 4 – Установка 7-го бита единицей id=65**

А в примере на рис.5 id =300, чтобы показать как будет изменяться число при превышении своего численного лимита.

```
unsigned char id = 300;
unsigned char mask = 1;
id = id | (mask << 6);
cout << int(id) << endl;
```



Консоль отладки Microsoft Visual Studio

108

**Рисунок 5 – Установка 7-го бита единицей id=300**

### **Задание 1.в.**

**Условие:** Реализуйте код листинга 1, объясните выводимый программой результат.

Рассмотрим код на рис.6 и напомним подобный данному, изобразив его на рис.7 с выводимыми результатами.

Код работает следующим образом: инициализируется число и вместе с ним маска, которая будет по нему проходить.

Маска приводится в старший разряд 32-ух битов для последующей проверки.

Во время проверки мы проходимся по всем разрядам проверяемого числа, начиная со старшего, постепенно продвигаясь к самому младшему.

$i = 1$ , потому что  $n = 32$ , а кол-во проходов цикла должно быть равно 32 ( $32 - 1 + 1 = 32$ ), что мы и получаем при данных условиях.

При этом маска уменьшает свой старший разряд, чтобы осуществить проверку всех разрядов рассматриваемого числа. И по окончании проверки мы выведем на экран рассматриваемое число в двоичном виде.

```

1 //Битовые операции
2 #include <cstdlib>
3 #include <iostream>
4 #include <Windows.h>
5 #include <bitset>
6 using namespace std;
7
8 int main()
9 {
10     SetConsoleCP(1251);
11     SetConsoleOutputCP(1251);
12
13     unsigned int x = 25;
14     const int n = sizeof(int)*8; //=32 - количество разрядов в числе типа int
15     unsigned maska = (1 << n - 1); //1 в старшем бите 32-разрядной сетки
16     cout << "Начальный вид маски: " << bitset<n>(maska) << endl;
17     cout << "Результат: ";
18     for (int i = 1; i <= n; i++) //32 раза - по количеству разрядов:
19     {
20         cout << ((x & maska) >> (n - i));
21         maska = maska >> 1; //смещение 1 в маске на разряд вправо
22     }
23     cout << endl;
24     system("pause");
25     return 0;
26 }

```

Рисунок 6 – Листинг из задания 1.в.

```

void main() {
    setlocale(LC_ALL, "Russian");
    /*unsigned char a = 255;
    unsigned char mask = 1;
    a = a & ~(mask << 4);
    cout << int(a) << endl;*/

    /*unsigned char id = 300;
    unsigned char mask = 1;
    id = id | (mask << 6);
    cout << int(id) << endl;*/

    unsigned int x = 25;
    const int n = sizeof(int) * 8; //=32 - кол-во разрядов в числе типа int
    unsigned mask = (1 << n - 1); //1 в старшем бите 32-разрядной сетки
    cout << "Начальный вид маски: " << bitset<n>(mask) << endl; // bitset - хранит битовые значения
    cout << "Результат: ";
    for (int i = 1; i <= n; i++) // 32 раза по кол-ву разрядов
    {
        cout << ((x & mask) >> (n - i));
        mask = mask >> 1; //смещение 1 в маске на разряд вправо
    }
    cout << endl
    << n << endl
    << sizeof(int);
}

```

Консоль отладки Microsoft Visual Studio

```

Начальный вид маски: 10000000000000000000000000000000
Результат: 0000000000000000000000000000000011001
32
4

```

Рисунок 7 – Реализованный из рис.6 код с полученными результатами

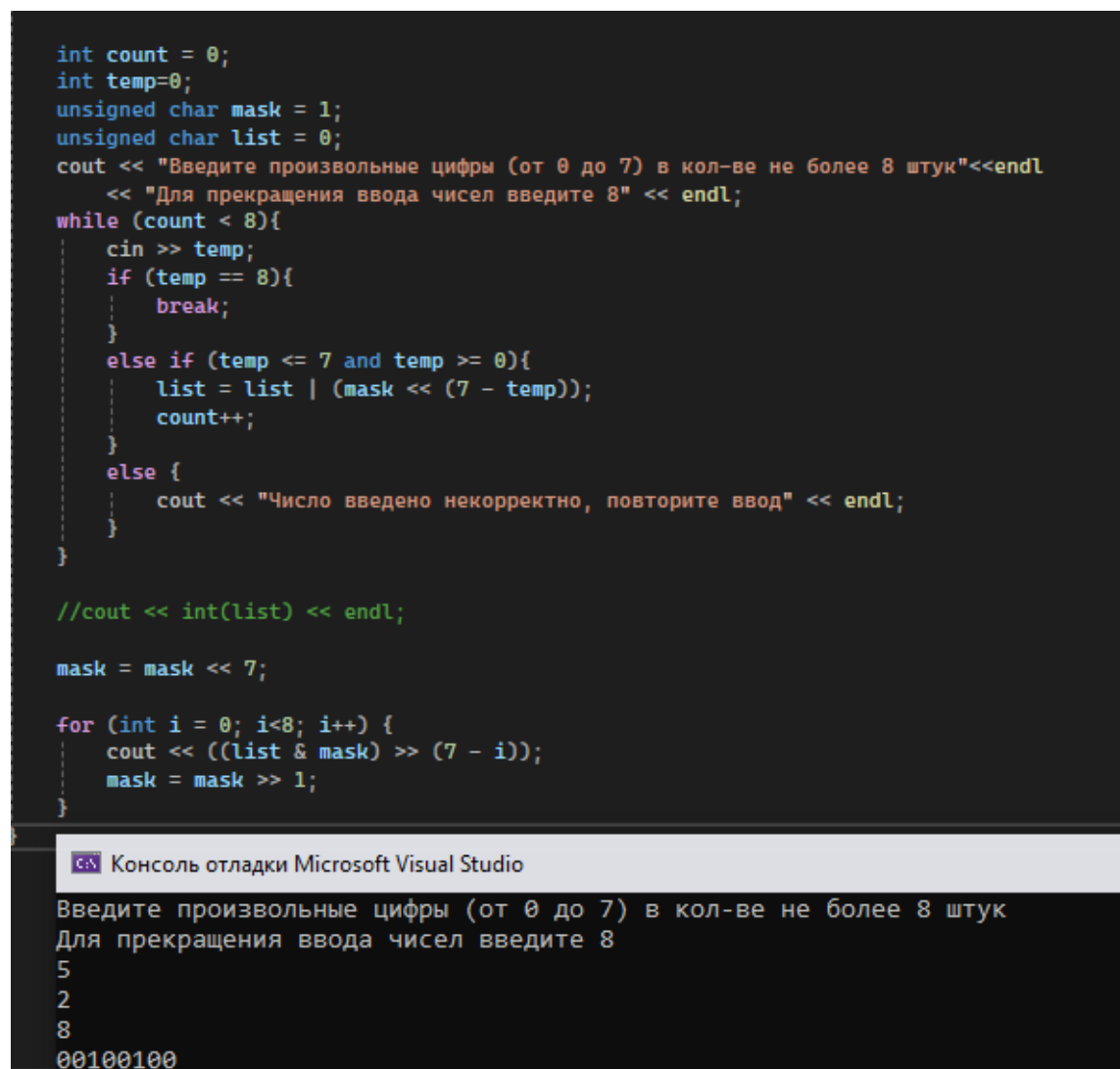
## ЗАДАНИЕ 2

### Задание 2.а.

**Условие:** 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. Если количество чисел в исходной последовательности больше 8 и/или значения превосходят 7, можно подобрать тип беззнакового числа для битового массива с подходящим размером разрядной сетки – до 64 в типе unsigned long long (см. табл. 1).

**Математическая модель:** создаём наш список в виде переменной типа unsigned char и маску, после чего вводим значения, проходящие через проверку, и полученный результат выводим на экран.

На рис. 7 предоставлен как сам код, так и его результат при вводе только 2 чисел.



```
int count = 0;
int temp=0;
unsigned char mask = 1;
unsigned char list = 0;
cout << "Введите произвольные цифры (от 0 до 7) в кол-ве не более 8 штук"<<endl
<< "Для прекращения ввода чисел введите 8" << endl;
while (count < 8){
    cin >> temp;
    if (temp == 8){
        break;
    }
    else if (temp <= 7 and temp >= 0){
        list = list | (mask << (7 - temp));
        count++;
    }
    else {
        cout << "Число введено некорректно, повторите ввод" << endl;
    }
}

//cout << int(list) << endl;

mask = mask << 7;

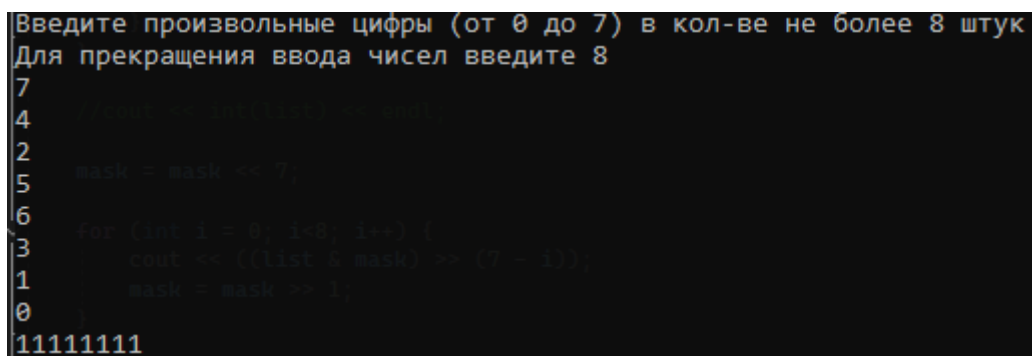
for (int i = 0; i<8; i++) {
    cout << ((list & mask) >> (7 - i));
    mask = mask >> 1;
}
```

Консоль отладки Microsoft Visual Studio

Введите произвольные цифры (от 0 до 7) в кол-ве не более 8 штук  
Для прекращения ввода чисел введите 8  
5  
2  
8  
00100100

Рисунок 8 – Код для задания 2.а

А на рис. 9 показан результат работы программы при вводе всех возможных чисел.



```
Введите произвольные цифры (от 0 до 7) в кол-ве не более 8 штук
Для прекращения ввода чисел введите 8
7
4
2
5
6
3
1
0
11111111
```

Рисунок 9 – Результат работы программы при вводе всех возможных чисел

### Задание 2.6

**Условие:** 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`.

Для решения задачи и последующего усовершенствования кода пришлось зайти совершенно с другой стороны: надо создать линейный взаимосвязный список при помощи структур.

В самом начале решения создадим маску типа `unsigned long long` и создадим всё необходимое для корректной работы программы, после чего приступим к введению различных чисел. Если полученные числа будут превосходить разрядную сетку, то тогда будут создаваться новые элементы структур с пустым ядром (числом) до тех пор, пока разрядная сетка не расширится на достаточный уровень. И в конце расширения сетки введённое число в неё добавится и, что самое главное, сохранится в ней.

Вышеописанный код представлен на рис. 10-11.

Также можно заметить, что на рис. 10 присутствуют лишние строчки кода в конце – это импровизированный вывод до 192 не включительно с условием того, что их до этого ввёл пользователь. И результаты работы данной программы можно наблюдать на рис. 12.

Все недостатки предыдущего метода было по большей степени устранены и программа приняла более усовершенствованный вид.

```

unsigned long long mask = 1;

Deck* ls = new Deck(0);
ls->make_last(ls);
ls->make_first(ls);
Deck* first = ls;

cout << "Введите желаемые числа (для прекращения работы ввода введите -1): " << endl;
while (true){
    int temp;
    cin >> temp;
    if (temp == -1)
        break;

    for (int i = 0; i < (temp / 64); i++) {
        if (ls->next == nullptr)
        {
            ls->next = new Deck(0);
        }

        ls = ls->next;
        ls->make_first(first);
    }

    temp %= 64;
    unsigned long long tempo = (ls->core | (mask << temp));
    ls->change_core(tempo);
    ls = ls->first;
}

for (int j = 0; j < 3; j++) {
    mask = mask << 63;
    for (int i = 0; i < 64; i++) {
        cout << (((ls->core) & mask) >> (63 - i));
        mask = mask >> 1;
    }
    mask = 1;
    cout << endl;
    ls = ls->next;
}

```

Рисунок 10 – 1-ая часть кода программы для задания 2.б.





```

unsigned char mask = 1;

Deck* ls = new Deck(0);
ls->make_last(ls);
ls->make_first(ls);
Deck* first = ls;

cout << "Введите желаемые числа (для прекращения работы ввода введите -1): " << endl;
while (true){
    int temp;
    cin >> temp;
    if (temp == -1)
        break;

    for (int i = 0; i < (temp / 8); i++) {
        if (ls->next == nullptr)
        {
            ls->next = new Deck(0);
        }

        ls = ls->next;
        ls->make_first(first);
    }

    temp %= 8;
    unsigned char tempo = (ls->core | (mask << temp));
    ls->change_core(tempo);
    ls = ls->first;
}

for (int j = 0; j < 3; j++) {
    mask = mask << 7;
    for (int i = 0; i < 8; i++) {
        cout << (((ls->core) & mask) >> (7 - i));
        mask = mask >> 1;
    }
    mask = 1;
    cout << endl;
    ls = ls->next;
}

```

**Рисунок 13 – 1-ая часть кода программы для задания 2.в.**

Можно заметить, что на рис. 15 число 5 повторяется, однако это совершенно никак не меняет конечного результата – число 5 введено, а значит, оно будет в нашем списке. Аналогично ко всем прочим числам.

```

struct Deck {
public:
    Deck* first = nullptr;
    Deck* last = nullptr;
    Deck* next = nullptr;
    unsigned char core = 0;

    Deck(unsigned char _current) : core{ _current } {
    };

    void add(Deck* next) {
        this->next = next;
    }

    void make_first(Deck* first)
    {
        this->first = first;
    }

    void make_last(Deck* last) {
        this->last = last;
    }

    void change_core(unsigned char core) {
        this->core = core;
    }
}

```

Рисунок 14 – 2-ая часть кода программы для задания 2.в.

```

Введите желаемые числа (для прекращения работы ввода введите -1):
8
5
7
5
12
5
23
-1
10100000
00010001
10000000

```

Рисунок 15 – Результат работы программы задания 2.в.

### Задание 3

#### Задание 3.а.

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

#### Задание 3.б.

**Условие:** Определите программно объём оперативной памяти, занимаемый битовым массивом.

Метод с конструктами является не самым эффективным, когда речь заходит о большом кол-ве данных, поэтому в данной программе будет использоваться обычный линейный список типа unsigned char.

Для начала мы создадим файл numList.txt, куда запишем значения в диапазоне от 0 включительно до  $10^7$  не включительно. Записав значения в файл, можно начать их считывание и последующую запись в наш список.

В программе также реализованы выходы времени сортировки файла и веса массива.

Код программы представлен на рис. 17.

```
ifstream reader;
reader.open("numList.txt");

ofstream writer("numList.txt");

for (int i = 0; i < pow(10, 7); i++)
{
    writer << rand() % tempor << endl;
}
writer.close();

unsigned char mask = 1;

int temp;
string st;
unsigned char* sp = new unsigned char[pow(10, 6) * 1.25];
int sec = time(NULL);
int id;
cout << sec << endl;
for (int k = 0; k < pow(10, 7); k++) {
    mask = 1;
    reader >> st;
    temp = 0;
    for (int i = 0; i < st.size(); i++)
    {
        temp += ((int)st[i] - 48) * pow(10, st.size() - i - 1);
    }

    id = temp / 8;
    sp[id] = sp[id] | (mask << (temp % 8));
}

cout << time(NULL) - sec << endl <<
    sizeof(sp) * pow(10, 6) * 1.25 / 1024 / 1024 / 8 << endl;
reader.close();
```

Рисунок 16 – Код работы программы для 3.а. и 3.б.

Также на рис. 17 отображён результат работы программы с затраченным на сортировку временем в секундах и веса массива.

Было затрачено 7 секунд на сортировку  
Массив весит 1.19209 Мбайт

**Рисунок 17 – Результат работы кода программы заданий 3.а. и 3.б.**

### **Задание 1 части 5.2**

**Условие:** Разработать программу поиска записей с заданным ключом в двоичном файле с применением различных алгоритмов.

#### **Задание 1.1**

**Условие:** 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.

Как и требуется в задании, создадим бинарный файл (рис.18). Можно было бы сначала создать текстовый файл, однако, разобравшись с бинарными файлами, я решил, что будет лучше создавать конечный вид файла без каких-либо промежуточных состояний.

```
int temp = rand() % 100000000 + 100000000;
wrBin.write((char*)&temp, sizeof(int));
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 5; j++) {
        sp[j] = (char)-52;
    }
    temp = rand() % 3 + 2;
    for (int j = 0; j < temp; j++)
    {
        if (sp[0] == (char)-52)
            sp[j] = (char)(rand() % 26 + 65);
        else
            sp[j] = (char)(rand() % 26 + 97);
    }
    wrBin.write(sp, sizeof(sp));
}
```

**Рисунок 18 – Создание бинарного файла**

#### **Задание 1.2**

**Условие:** Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей. Составить таблицу с указанием результатов замера времени.

Разработать программу линейного поиска было не сложно – всё-таки это самый примитивный способ поиска значений (рис. 19).

```

int goal = 30000000;

auto start = chrono::high_resolution_clock::now();
for (int i = 0; i < 100; i++) {
    rBin.read((char*)&temp, sizeof(int));
    if (temp == goal)
        break;
    for (int j = 0; j < 4; j++)
        rBin.read(sp, sizeof(sp));
}
auto end = chrono::high_resolution_clock::now();

chrono::duration<double> time = end - start;

cout << "Найдено " << temp << endl
     << "За " << time.count() << " секунд";

```

**Рисунок 19 – Программа линейного поиска**

Теперь же начнём экспериментировать со значениями от 100 до 10000 записей и отобразим соответствующие результаты на рис. 20-22.

```

Найдено 30000000
За 7.01e-05 секунд

```

**Рисунок 20 – Результат работы с 100 значений**

```

Найдено 30000000
За 0.0010156 секунд

```

**Рисунок 21 – Результат работы с 1000 значений**

```

Найдено 30000000
За 0.0060442 секунд

```

**Рисунок 22 – Результат работы с 10000 значений**

Из полученных результатов в таблице 1 можно заметить, что время увеличивается линейно в зависимости от кол-ва переменных.

**Таблица 1 – Результаты замера времени при работе с разл-ым кол-вом пер-ых**

Кол-во переменных	Время, с
100	$7 \cdot 10^{-5}$
1000	0.001
10000	0.006

### Задание 1.3

**Условие:** Для оптимизации поиска в файле создать в оперативной памяти структур данных – таблицу, содержащую ключ и ссылку (смещение) на запись в файле. 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. Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. Возвращает прочитанную запись как результат. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей. Составить таблицу с указанием результатов замера времени.

В задании требуется сделать таблицу, однако же данное понятие можно упростить до одномерного массива, где индексы элементов – это ссылки на место в файле, а сами элементы – это ключи. В реализации данного способа решения задачи можно ограничиться всего лишь одним новым полем, а именно массивом целочисленных значений **ar**.

В следующем же задании требуется отсортированный массив, поэтому придётся добавить ещё один массив **arLink**, который будет хранить ссылки на соответствующие элементы из **ar**. Реализация требуемой функции представлена на рис. 23.

```
int searchLink(int* ar, int* arL, int size, int key) {
    int tempS = size/2;
    int prev = tempS;
    int tempPrev;
    if (ar[tempS] == key)
        return arL[tempS];
    else if (ar[tempS] > key)
        tempS -= tempS / 2;
    else
        tempS += (size - tempS) / 2;
    while (true) {
        tempPrev = tempS;
        if (ar[tempS] == key)
            return arL[tempS];
        else if (ar[tempS] > key){
            tempS -= abs(tempS - prev) / 2;
            if (tempPrev == prev)
                tempS--;
        }
        else{
            tempS += abs(tempS - prev) / 2;
            if (tempPrev == prev)
                tempS++;
        }
        prev = tempPrev;
    }
    return -1;
}
```

Рисунок 23 – Реализация функции бинарного поиска ссылки по ключу

Теперь же разработаем функцию поиска ключа по ссылке, делается это нетрудно, а вид полученной функции отображён на рис.24.

```
int poisk = 90;
rBin.seekg(poisk * (sizeof(int) + sizeof(sp) * 4));
rBin.read((char*)&temp, sizeof(int));
cout << temp << endl
      << arK[poisk] << endl;
```

**Рисунок 24 – Функция поиска ключа по ссылке**

Далее мы определим время, затрачиваемое на работу с 100, 1000 и 10000 переменными в крайних случаях в таблице 2.

Таблица 2 – Время работы с разл-ым кол-вом пер-ых.

Кол-во переменных	Поиск ссылки по ключу, с	Поиск ключа по ссылке, с
100	$1 \cdot 10^{-7}$	$8.6 \cdot 10^{-6}$
1000	$1.5 \cdot 10^{-7}$	$1,1 \cdot 10^{-5}$
10000	$3 \cdot 10^{-7}$	$1.8 \cdot 10^{-5}$

Как можно заметить из таблицы, оба способа сортировки получились достаточно эффективными.

## ВЫВОД

В 5.1 познакомились с битовой сортировкой значений, а также инструментами осуществляющими данную сортировку. Программно определили время, затрачиваемое на сортировку  $10^7$  чисел и изучили эффективный метод сортировки.

В 5.2 ознакомились с бинарными файлами, изучили работу с ними: запись, хранение и изменение в них данных. Кроме того опробовали различные методы нахождения разных переменных отличающимися друг от друга способами.

## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

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).