



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных  
технологий

## **Отчет по практической работе №1**

по дисциплине «Структуры и алгоритмы обработки данных»  
по теме «Поразрядные операции»

**Выполнил:**

Студент группы ИКБО-13-22

Лещенко Вячеслав Романович

**Проверил:**

ассистент Муравьёва Е.А.

МОСКВА 2023 г.

# Практическая работа № 1

## Цель работы

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

## Ход работы

### Вариант 20

20	5-ый и 7-ой справа	С 7-ого четыре бита слева	16	8	Установить n-ый бит в 1, используя маску (var 1)
----	-----------------------	---------------------------------	----	---	---

### Задание 1

#### Формулировка задачи:

1) Определить переменную целого типа, присвоить ей значение, используя константу в шестнадцатеричной системе счисления. Разработать оператор присваивания и его выражение, которое установит заданные в задании биты исходного значения переменной в значение 1, используя соответствующую маску и поразрядную операцию.

#### Решение:

- Для выполнения данной задачи нам задается число в 2-ой системе.
- Для установки 5-го и 7-го битов этого числа в единицу побитно сложим его с данной маской: (0000000010100000) и получим нужное число.
- Полученное число выводим в консоль.

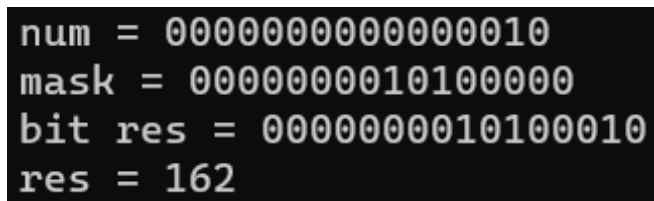
```

unsigned short int num = 0;
unsigned short int mask = 0;

num = 0b10;
mask = 0b00000000010100000;
io.output("num = ", "");
io.output(std::bitset<16>(num).to_string());
io.output("mask = ", "");
io.output(std::bitset<16>(mask).to_string());
io.output("bit res = ", "");
num |= mask;
io.output(std::bitset<16>(num).to_string());
io.output("res = ", "");
io.output(num);

```

Листинг 1 – код задачи 1.1



```

num = 00000000000000010
mask = 0000000010100000
bit res = 0000000010100010
res = 162

```

Рисунок 1 – результат тестирования 1.1

2) Определить переменную целого типа. Разработать оператор присваивания и его выражение, которое обнуляет заданные в задании биты исходного значения переменной, используя соответствующую маску и поразрядную операцию. Значение в переменную вводится с клавиатуры.

#### Решение:

- Для выполнения данной задачи нам вводится число в 10-ой системе, которое переводится в 2-ую 16 битную систему.
- Для установки 4-ёх битов слева после 7-го этого числа в нули побитно умножим его с данной маской: (111110000111111) и получим нужное число.
- Полученное число выводим в консоль.

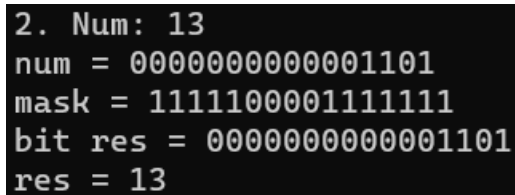
```

unsigned short int num = 0;
unsigned short int mask = 0;

num = io.input<unsigned short int>("2. Num: ");
mask = 0b1111100001111111;
io.output("num = ", "");
io.output(std::bitset<16>(num).to_string());
io.output("mask = ", "");
io.output(std::bitset<16>(mask).to_string());
io.output("bit res = ", "");
num &= mask;
io.output(std::bitset<16>(num).to_string());
io.output("res = ", "");
io.output(num);

```

Листинг 2 – код задачи 1.2



```

2. Num: 13
num = 000000000000001101
mask = 1111100001111111
bit res = 000000000000001101
res = 13

```

Рисунок 2 – результат тестирования 1.2

3) Определить переменную целого типа. Разработать оператор присваивания и выражение, которое умножает значение переменной на число, указанное в третьем столбце варианта, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.

#### Решение:

- Для выполнения данной задачи нам вводится число в 10-ой системе, которое переводится в 2-ую 16 битную систему.
- Для увеличения **в 8 раз** этого числа произведем побитный сдвиг в лево на 3 (степень 2 для получения числа 8).
- Полученное число выводим в консоль.

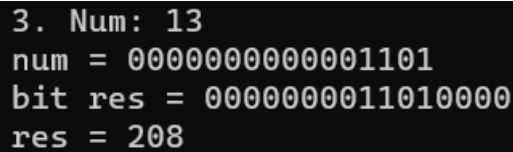
```

unsigned short int num = 0;

num = io.input<unsigned short int>("3. Num: ");
io.output("num = ", "");
io.output(std::bitset<16>(num).to_string());
num <= 4;
io.output("bit res = ", "");
io.output(std::bitset<16>(num).to_string());
io.output("res = ", "");
io.output(num);

```

Листинг 3 – код задачи 1.3



```

3. Num: 13
num = 00000000000001101
bit res = 0000000011010000
res = 208

```

Рисунок 3 – результат тестирования 1.3

4) Определить переменную целого типа. Разработать оператор присваивания и выражение, которое делит значение переменной на число, указанное в четвертом столбце варианта, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.

#### Решение:

- Для выполнения данной задачи нам вводится число в 10-ой системе, которое переводится в 2-ую 16 битную систему.
- Для уменьшения в 16 раз этого числа произведем побитный сдвиг в право на 4 (степень 2 для получения числа 16).
- Полученное число выводим в консоль.

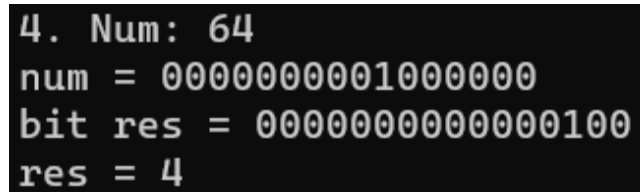
```

unsigned short int num = 0;

num = io.input<unsigned short int>("4. Num: ");
io.output("num = ", "");
io.output(std::bitset<16>(num).to_string());
num >>= 4;
io.output("bit res = ", "");
io.output(std::bitset<16>(num).to_string());
io.output("res = ", "");
io.output(num);

```

Листинг 4 – код задачи 1.4



```

4. Num: 64
num = 00000000001000000
bit res = 00000000000000100
res = 4

```

Рисунок 4 – результат тестирования 1.4

5) Определить переменную целого типа. Разработать оператор присваивания и выражение, в котором используются только поразрядные операции. В выражении используется маска – переменная. Маска может быть инициализирована единицей в младшем разряде (вар 1) или единицей в старшем разряде (вар 2). Изменяемое число вводится с клавиатуры.

#### Решение:

- Для выполнения данной задачи нам вводится число в 10-ой системе, которое переводится в 2-ую 16 битную систему и выбирается бит для перестановки в единицу.
- Для установки выбранного бита в 1 сложим его с данной маской: (00000000000000001) которую мы сдвинем влево на номер выбранного бита и получим нужное число.
- Полученное число выводим в консоль.

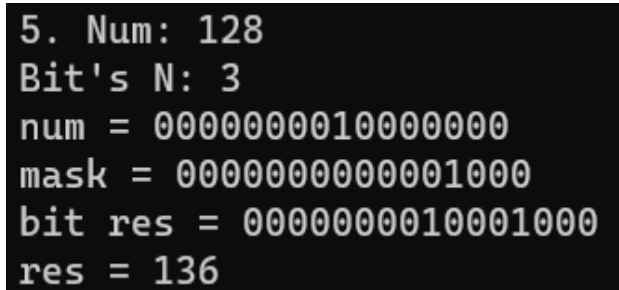
```

unsigned short int num = 0;
unsigned short int mask = 0;
unsigned short int bit_n = 0;

num = io.input<unsigned int>("5. Num: ");
bit_n = io.input<unsigned int>("Bit's N: ");
io.output("num = ", "");
io.output(std::bitset<16>(num).to_string());
io.output("mask = ", "");
io.output(std::bitset<16>(mask).to_string());
mask = 0b1 << bit_n;
num &= mask;
io.output("bit res = ", "");
io.output(std::bitset<16>(num).to_string());
io.output("res = ", "");
io.output(num);

```

Листинг 5 – код задачи 1.5



```

5. Num: 128
Bit's N: 3
num = 0000000010000000
mask = 0000000000001000
bit res = 0000000010001000
res = 136

```

Рисунок 5 – результат тестирования 1.5

## Задание 2

Формулировка задачи:

1) Реализовать 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.

**Решение:**

- Вводится n чисел, и устанавливаются соответствующие биты.
- На основе этого получаем двоичный вектор, который позволяет по нему отследить какие есть числа и отсортировать массив.

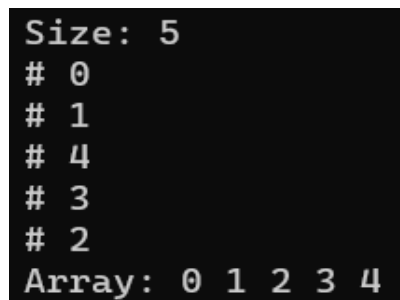
```

void bitwise_two()
{
    size_t size = io.input<size_t>("Size: ");
    unsigned char arr;
    unsigned short int bit;

    for (size_t i = 0; i < size; i++)
    {
        bit = io.input<unsigned short int>("# ");
        arr |= 0b1 << bit;
    }
    io.output("Array: ", "");
    for (size_t i = 0; i < size; i++)
    {
        if ((arr & (0b1 << i)) == (0b1 << i))
        {
            io.output(i, " ");
        }
    }
}

```

Листинг 6 – код задачи 2.1



```

Size: 5
# 0
# 1
# 4
# 3
# 2
Array: 0 1 2 3 4

```

Рисунок 6 – результат тестирования 2.1

2) 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.

#### Решение:

- Было принято решение создать собственный класс с удобными функциями и операторами.
- Индекс нужного нам бита преобразуется в индекс байта и позицию внутри байта для доступа к нужному биту.



```

class bitarray
{
private:
    unsigned char *arr;
    size_t size;
    size_t bytes;

public:
    bitarray(size_t size){
        this->size = size;
        this->bytes = (size_t) std::ceil(size / 8.);
        arr = new unsigned char[this->bytes];
        for (size_t i = 0; i < this->bytes; i++)
            arr[i] = 0;
    }

    ~bitarray() {
        delete[] arr;
    }

    size_t getsize()
    {
        return this->size;
    }

    void setbit(size_t index)
    {
        arr[(size_t) (index) / 8] |= 0b1 << (index % 8);
    }

    void resetbit(size_t index)
    {
        arr[(size_t) (index) / 8] &= ~(0b1 << (index % 8));
    }

    bool operator[](size_t index)
    {
        return (arr[(size_t) (index) / 8] & 0b1 << (index % 8)) == (0b1 <<
(index % 8));
    }
};

void bitwise_two_point_one()
{
    size_t size = io.input<size_t>("Size: ");
    bitarray arr(size);
    unsigned short int bit;

    for (size_t i = 0; i < size; i++)
    {
        arr.setbit(io.input<unsigned short int>("# "));
    }
    io.output("Array: ", "");
    for (size_t i = 0; i < size; i++)
    {
        if (arr[i])
        {
            io.output(i, " ");
        }
    }
}

```

Листинг 7 – код задачи 2.2

```
Size: 16  
15 11 10 12 1 3 9 8 13 14 7 4 2 5 0 6  
Array: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Рисунок 7 – результат тестирования 2.2

### 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. В отчёт внесите результаты тестирования для наибольшего количества входных чисел, соответствующего битовому массиву длиной 1МБ и программно определить объём оперативной памяти, занимаемый битовым массивом и время выполнения работы программы (сортировки) для каждого случая.

```

void bitwise_three()
{
    FIO fio("", "bita");

    srand(time(0));

    size_t len = fio.input<size_t>("Enter lenght: ");

    for (size_t i = 0; i < len; i++)
    {
        fio.output((((rand() + rand()) << 16) + (rand() + rand())) % len);
    }

    fio.output("File is ready.");

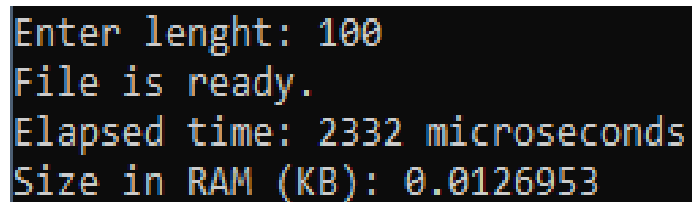
    fio.close();

    fio.set_in("bita");
    fio.set_out("bitb");
    fio.open();

    int var;
    bitarray array(len);
    measure(
        for (; !fio.is_end(); )
        {
            var = fio.input<int>();
            if (!fio.was_error())
                array.setbit(var);
            fio.clear_error();
        }
        for (size_t i = 0; i < array.getsize(); i++)
        {
            if (array[i])
                fio.output(i);
        }, "Elapsed time: "
    );
    fio.output("Size in RAM (KB): ", "");
    fio.output(array.getbytes() / 1024.);
}

```

Листинг 8 – код задачи 3.1

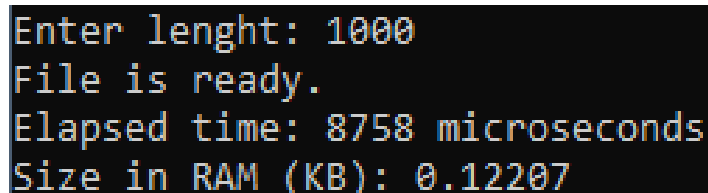


```

Enter lenght: 100
File is ready.
Elapsed time: 2332 microseconds
Size in RAM (KB): 0.0126953

```

Рисунок 8 – результат тестирования для 100 элементов 3.1

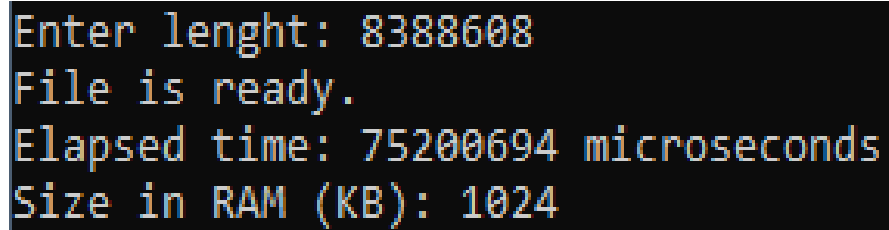


```

Enter lenght: 1000
File is ready.
Elapsed time: 8758 microseconds
Size in RAM (KB): 0.12207

```

Рисунок 9 – результат тестирования для 1000 элементов 3.1



```
Enter lenght: 8388608
File is ready.
Elapsed time: 75200694 microseconds
Size in RAM (KB): 1024
```

Рисунок 10 – результат тестирования для 8388608 элементов 3.1

### **Вывод**

Был практически освоен принцип битовой сортировки. Данная сортировка подходит для создания упорядоченных массивов уникальных элементов, так как она не учитывает повторения. Сортировка довольно быстрая, но сложно понять как она работает “под капотом”, хотя базовый принцип совмещения индексации и двоичного переключения состояния интуитивно понятен.