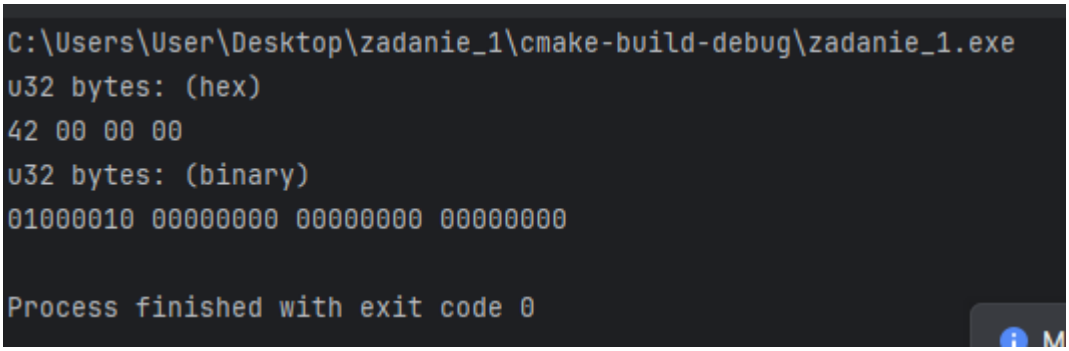
Задание №1

Листинг решения

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cstdint>

using namespace std;

// Функция для преобразования значения nibble (0-15) в шестнадцатеричный
символ
char nibble_to_hex(uint8_t i) {
    assert(0x0 <= i && i <= 0xf);
    char digits[] = "0123456789abcdef";
    return digits[i];
}

// Функция для печати одного байта в шестнадцатеричном представлении
void print_in_hex(uint8_t byte) {
    cout << nibble_to_hex(byte >> 4) << nibble_to_hex(byte & 0xf);
}

// Функция для преобразования указателя void* в указатель на байты uint8_t*
const uint8_t *as_bytes(const void *data) {
    return reinterpret_cast<const uint8_t *>(data);
}

// Функция для печати блока данных в шестнадцатеричном представлении
void print_in_hex(const void *data, size_t size) {
    const uint8_t *bytes = as_bytes(data);
    for (size_t i = 0; i < size; i++) {
        print_in_hex(bytes[i]);
        if ((i + 1) % 16 == 0) {
            cout << '\n';
        } else {
            cout << ' ';
        }
    }
}

// Функция для получения бита с номером bit из байта byte
char bit_digit(uint8_t byte, uint8_t bit) {
    if (byte & (0x1 << bit)) {
        return '1';
    }
    return '0';
}

// Функция для печати одного байта в двоичном представлении
void print_in_binary(uint8_t byte) {
    for (uint8_t bit = 7; bit < 8; bit--) {
        cout << bit_digit(byte, bit);
    }
```

```
}

// Функция для печати блока данных в двоичном представлении
void print_in_binary(const void *data, size_t size) {
    const uint8_t *bytes = as_bytes(data);
    for (size_t i = 0; i < size; i++) {
        print_in_binary(bytes[i]);
        if ((i + 1) % 4 == 0) {
            cout << '\n';
        } else {
            cout << ' ';
        }
    }
}

int main() {
    uint32_t u32 = 0x42;
    cout << "u32 bytes: (hex)\n";
    print_in_hex(&u32, sizeof(u32));
    cout << "\nu32 bytes: (binary)\n";
    print_in_binary(&u32, sizeof(u32));
    return 0;
}
```

Результат



Задание №2

Листинг решения

```
#include <iostream>
#include <iomanip>
#include <cstdint>
#include <bitset>
#include <sstream>
#include <regex>

using namespace std;

// Функция для печати числа типа uint16_t в шестнадцатеричном и двоичном виде
string number_to_string(uint16_t num) {
    ostringstream oss;
    oss << setfill('0') << setw(2) << hex << static_cast<uint8_t>(num >> 8)
<< ' '
        << setfill('0') << setw(2) << hex << static_cast<uint8_t>(num);
    string hex_str = oss.str();
    oss.str("");
    oss << bitset<8>(num >> 8) << ' ' <<
bitset<8>(static_cast<uint8_t>(num));
    string bin_str = oss.str();
    return hex_str + " " + bin_str;
}

int main() {
    uint16_t num1, num2; // Два операнда
    char op; // Оператор
    // Ввод выражения в формате "num1 op num2"
    cin >> dec >> num1 >> op >> num2;

    uint16_t result; // Результат операции

    // Выполнение операции в зависимости от введенного оператора
    switch (op) {
        case '&':
            result = num1 & num2;
            break;
        case '|':
            result = num1 | num2;
            break;
        case '^':
            result = num1 ^ num2;
            break;
        default:
            cout << "Unknown operator\n";
            return 1;
    }

    // Формирование строки вывода
    string output = number_to_string(num1) + " & " + number_to_string(num2) +
                    " = " + number_to_string(result);

    // Замена "&" на соответствующий оператор
```

```
    if (op == '|') {
        output = regex_replace(output, regex("&"), "|");
    } else if (op == '^') {
        output = regex_replace(output, regex("&"), "^");
    }

    // Печать результата
    cout << output << '\n';

    return 0;
}
```

Результат

```
C:\Users\User\Desktop\zadanie2\cmake-build-debug\zadanie2.exe
3 & 2
0  0♥ 00000000 00000011 & 0  0☻ 00000000 00000010 = 0  0☻ 00000000 00000010

Process finished with exit code 0
```

Задание №3

Листинг решения

```
#include <iostream>
#include <iomanip>
#include <cstdint>
#include <cstring>
#include <bitset>

using namespace std;

// Структура Student, описывающая студента
struct Student {
    char name[17];            // Имя (массив из 17 символов, включая
завершающий '\0')
    uint16_t year;            // Год поступления (беззнаковое целое, 2 байта)
    float average_score;      // Средний балл (с плавающей запятой)
    uint8_t gender : 1;       // Пол, представленный одним битом (0 —
женский, 1 — мужской)
    uint8_t courses_passed;   // Количество пройденных курсов
    Student* group_leader;    // Указатель на структуру Student, описывающую
старосту группы (для старосты — нулевой указатель)
};

int main() {
    // Объявление массива из трех структур Student
    Student students[3];

    // Заполнение массива данными о студентах
    strcpy(students[0].name, "Игорь");
    students[0].year = 2022;
    students[0].average_score = 3.38f;
    students[0].gender = 1;
    students[0].courses_passed = 1;
    students[0].group_leader = &students[2];

    strcpy(students[1].name, "Анна");
    students[1].year = 2021;
    students[1].average_score = 4.75f;
    students[1].gender = 0;
    students[1].courses_passed = 3;
    students[1].group_leader = &students[2];

    strcpy(students[2].name, "Мария");
    students[2].year = 2020;
    students[2].average_score = 4.92f;
    students[2].gender = 0;
    students[2].courses_passed = 5;
    students[2].group_leader = nullptr;

    // Печать адреса и размера массива
    cout << "Address of array: " << &students << ", size: " <<
sizeof(students) << " bytes\n\n";

    // Печать адресов и размеров всех элементов массива
```

```
    for (int i = 0; i < 3; i++) {
        cout << "Address of students[" << i << "]: " << &students[i]
             << ", size: " << sizeof(students[i]) << " bytes\n";
    }
    cout << '\n';

    // Печать информации о полях структуры для студента Игоря
    cout << "Information about fields of students[0] (Игорь):\n";
    cout << "Name: address: " << &students[0].name
         << ", offset: " << offsetof(Student, name)
         << ", size: " << sizeof(students[0].name) << " bytes\n";
    cout << "Year: address: " << &students[0].year
         << ", offset: " << offsetof(Student, year)
         << ", size: " << sizeof(students[0].year) << " bytes"
         << ", hex: " << hex << setw(4) << setfill('0') << students[0].year
         << ", bin: " << bitset<16>(students[0].year) << '\n';
    cout << "Average score: address: " << &students[0].average_score
         << ", offset: " << offsetof(Student, average_score)
         << ", size: " << sizeof(students[0].average_score) << " bytes\n";
    cout << "Courses passed: address: " <<
reinterpret_cast<void*>(&students[0].courses_passed)
         << ", offset: " << offsetof(Student, courses_passed)
         << ", size: " << sizeof(students[0].courses_passed) << " bytes"
         << ", hex: " << hex << static_cast<int>(students[0].courses_passed)
         << ", bin: " << bitset<8>(students[0].courses_passed) << '\n';
    cout << "Group leader: address: " << &students[0].group_leader
         << ", offset: " << offsetof(Student, group_leader)
         << ", size: " << sizeof(students[0].group_leader) << " bytes\n\n";

    // Печать всех элементов массива в шестнадцатеричном виде
    cout << "Array elements in hex:\n";
    for (int i = 0; i < 3; i++) {
        cout << "students[" << i << "]:\n";
        const uint8_t* bytes = reinterpret_cast<const
uint8_t*>(&students[i]);
        for (size_t j = 0; j < sizeof(Student); j++) {
            cout << hex << setw(2) << setfill('0') <<
static_cast<int>(bytes[j]) << ' ';
            if ((j + 1) % 16 == 0) {
                cout << '\n';
            }
        }
        cout << '\n';
    }

    return 0;
}
```

Результат

```
C:\Users\User\Desktop\zadanie3\cmake-build-debug\zadanie3.exe
Address of array: 0x4707bffd90, size: 120 bytes

Address of students[0]: 0x4707bffd90, size: 40 bytes
Address of students[1]: 0x4707bffdb8, size: 40 bytes
Address of students[2]: 0x4707bffde0, size: 40 bytes

Information about fields of students[0] (╨ÿ╨╡╨╡╤Ç╤î):
Name: address: 0x4707bffd90, offset: 0, size: 17 bytes
Year: address: 0x4707bffda2, offset: 18, size: 2 bytes, hex: 07e6, bin: 0000011111100110
Average score: address: 0x4707bffda4, offset: 14, size: 4 bytes
Courses passed: address: 0x4707bffda9, offset: 19, size: 1 bytes, hex: 1, bin: 00000001
Group leader: address: 0x4707bffdb0, offset: 20, size: 8 bytes

Array elements in hex:
students[0]:
d0 98 d0 b3 d0 be d1 80 d1 8c 00 00 00 00 00 00
00 00 e6 07 ec 51 58 40 9b 01 7b ae f6 7f 00 00
e0 fd bf 07 47 00 00 00
students[1]:
d0 90 d0 bd d0 bd d0 b0 00 00 00 00 00 00 00 00
00 00 e5 07 00 00 98 40 10 03 bf 07 47 00 00 00
e0 fd bf 07 47 00 00 00
students[2]:
d0 9c d0 b0 d1 80 d0 b8 d1 8f 00 ae f6 7f 00 00
80 16 e4 07 a4 70 9d 40 00 05 00 00 00 00 00 00
00 00 00 00 00 00 00 00

Process finished with exit code 0
```

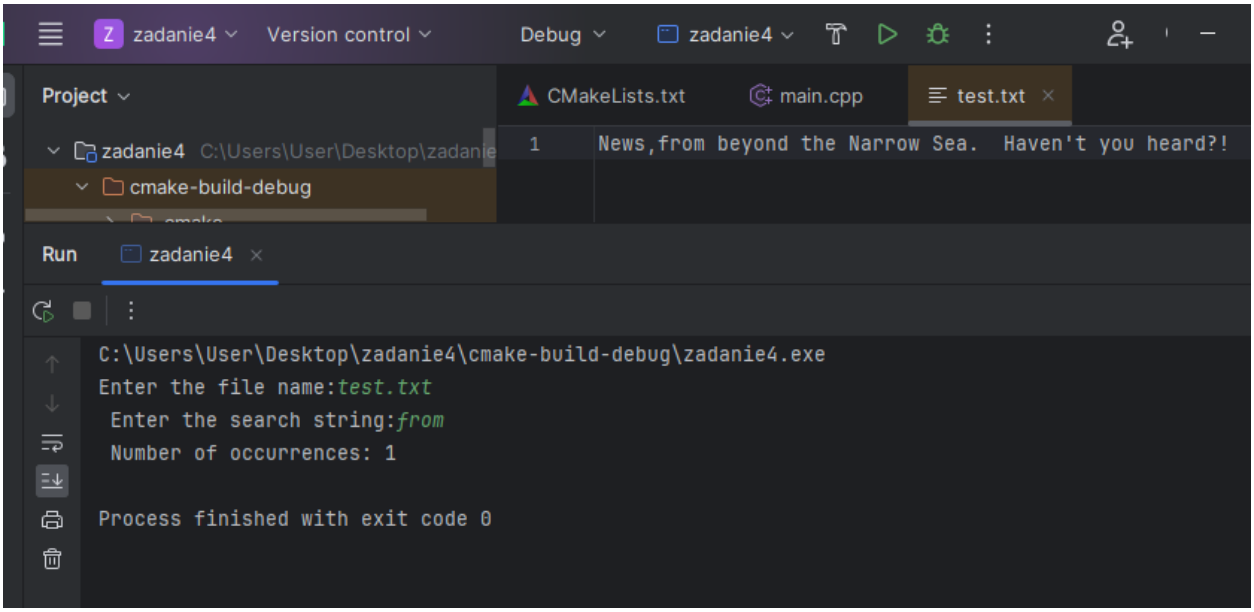
Задание №4

Листинг решения

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <cctype>

using namespace std;

// Функция для проверки корректности имени файла
bool is_valid_filename(const char* filename) {
    const char* invalid_chars = "\"<>|";
    const size_t filename_length = strlen(filename);

    // Проверка на наличие запрещенных символов
    if (strpbrk(filename, invalid_chars) != nullptr) {
        return false;
    }

    // Проверка правильности использования двоеточия
    const char* colon = strchr(filename, ':');
    if (colon != nullptr) {
        if (colon != filename + 1 || !isalpha(filename[0]) || colon[1] !=
'\\') {
            return false;
        }
    }

    // Проверка расширения файла
    const char* extension = strrchr(filename, '.');
    if (extension != nullptr) {
        if (strcmp(extension, ".txt") != 0 && strcmp(extension, ".TXT") != 0)
{
            return false;
        }
    }

    return true;
}

int main() {
    char filename[256];

    // Запрос имени файла у пользователя
    cout << "Enter the file name: ";
    cin.getline(filename, sizeof(filename));

    // Проверка корректности имени файла
    if (!is_valid_filename(filename)) {
        cout << "Invalid file name!" << endl;
        return 1;
    }

    // Добавление расширения .txt, если его нет
```

```
    if (strrchr(filename, '.') == nullptr) {
        strcat(filename, ".txt");
    }

    // Открытие файла для чтения
    ifstream file(filename, ios::binary);
    if (!file) {
        cout << "Failed to open the file!" << endl;
        return 1;
    }

    // Определение размера файла
    file.seekg(0, ios::end);
    streamsize file_size = file.tellg();
    file.seekg(0, ios::beg);

    // Выделение памяти для хранения содержимого файла
    char* file_content = new char[file_size];

    // Чтение содержимого файла в память
    file.read(file_content, file_size);
    file.close();

    // Запрос строки у пользователя
    char search_string[256];
    cout << "Enter the search string: ";
    cin.getline(search_string, sizeof(search_string));

    // Подсчет количества вхождений строки в текст файла
    size_t count = 0;
    const char* match = strstr(file_content, search_string);
    while (match != nullptr) {
        count++;
        match = strstr(match + 1, search_string);
    }

    // Вывод результата
    cout << "Number of occurrences: " << count << endl;

    // Освобождение выделенной памяти
    delete[] file_content;

    return 0;
}
```

Результат


zadanie4
Version control
Debug
zadanie4
Project
zadanie4 C:\Users\User\Desktop\zadanie
cmake-build-debug
CMakeLists.txt
main.cpp
test.txt
1
News,from beyond the Narrow Sea. Haven't you heard?!
Run
zadanie4
C:\Users\User\Desktop\zadanie4\cmake-build-debug\zadanie4.exe
Enter the file name:test.txt
Enter the search string:from
Number of occurrences: 1
Process finished with exit code 0

Контрольные вопросы

1. Операторы new и delete используются для динамического выделения и освобождения памяти. При удалении массива используются квадратные скобки (delete[]), а при удалении одиночного значения - просто delete.
2. Для работы с квадратной матрицей динамически выделяется массив указателей на строки, а затем для каждой строки выделяется динамический массив элементов.
3. "Рваный" массив - это массив массивов разной длины. Память под элементы выделяется отдельно для каждого подмассива, доступ к элементам осуществляется через двойную индексацию.
4. Арифметика с плавающей запятой не является точной из-за ограниченной точности представления чисел. Сравнение значений на равенство может давать неожиданные результаты.
5. Для корректного сравнения значений с плавающей запятой используется сравнение с допустимой погрешностью (эпсилон). Для борьбы с ошибками округления применяются специальные методы, такие как использование чисел с фиксированной запятой.
6. Арифметика с фиксированной запятой использует целые числа для представления дробных значений. Преимущества: точность, детерминированность. Недостатки: ограниченный диапазон, необходимость масштабирования.
7. Оператор reinterpret_cast позволяет переинтерпретировать биты одного типа данных как другой тип. Удобен для низкоуровневых операций, но может привести к неопределенному поведению при неправильном использовании.
8. Выравнивание данных - это требование процессора к размещению данных в памяти по определенным адресам. Необходимо для

оптимизации доступа к памяти. Контролируется с помощью выравнивающих спецификаторов (alignas) или директив компилятора.

9. Размер переменной определяется оператором sizeof. Он может отличаться от размера полезных данных из-за выравнивания или дополнительных служебных данных (например, в случае классов с виртуальными функциями).
10. Побитовые операторы в C++: & (И), | (ИЛИ), ^ (исключающее ИЛИ), ~ (НЕ), << (сдвиг влево), >> (сдвиг вправо). Пример: 0b1010 & 0b1100 = 0b1000.
11. Битовые флаги - это отдельные биты, используемые для хранения состояний (вкл/выкл). Битовые маски - это шаблоны битов для выделения нужных флагов. Числа в разных системах счисления записываются с префиксами: 0b (двоичная), 0 (восьмеричная), 0x (шестнадцатеричная).
12. а) uint8_t, uint16_t, uint32_t; б) std::bitset<N>; в) bit-поля в структурах (uint32_t field : 9;).
13. Массивы объявляются с указанием типа и размера: int arr[10];. Инициализация - в фигурных скобках: {1, 2, 3}. Доступ к элементам - через индексы в квадратных скобках: arr[0]. Размер определяется оператором sizeof.
14. std::array - шаблонный класс для представления массивов фиксированного размера. Преимущества: проверка выхода за границы, удобные методы (size(), fill() и др.), совместимость с STL-алгоритмами.
15. Строки в стиле C - это массивы символов, заканчивающиеся нулевым символом ('\0'). Для работы с ними используются функции из <cstring>: strlen(), strcpy(), strcat(), strcmp() и др.