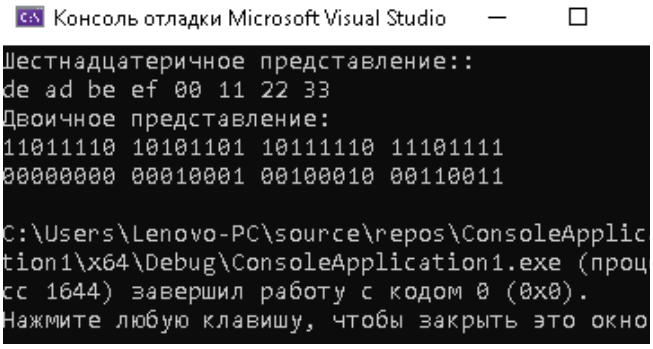
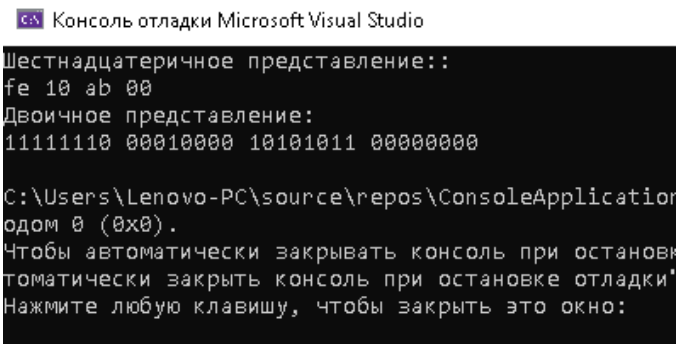
Задание на лабораторную работу
Общее задание
№1

main.cpp

```cpp
#include <iostream>
#include <iomanip>
#include <bitset>

void print_in_hex(uint8_t byte) { // Функция для печати одного байта в шестнадцатеричном представлении
    std::cout << std::hex << std::setw(2) << std::setfill('0') << static_cast<int>(byte);
}

void print_in_hex(const void* data, size_t size) { // Функция для печати блока данных в шестнадцатеричном представлении
    const uint8_t* byteData = static_cast<const uint8_t*>(data);
    for (size_t i = 0; i < size; ++i) {
        if (i > 0 && i % 16 == 0) std::cout << std::endl;
        print_in_hex(byteData[i]);
        std::cout << " ";
    }
    std::cout << std::endl;
}

void print_in_binary(uint8_t byte) { // Функция для печати одного байта в двоичном представлении
    std::cout << std::bitset<8>(byte);
}

void print_in_binary(const void* data, size_t size) { // Функция для печати блока данных в двоичном представлении
    const uint8_t* byteData = static_cast<const uint8_t*>(data);
    for (size_t i = 0; i < size; ++i) {
        if (i > 0 && i % 4 == 0) std::cout << std::endl;
        print_in_binary(byteData[i]);
        std::cout << " ";
    }
    std::cout << std::endl;
}

int main() {
    setlocale(LC_ALL, "RUS");

    uint8_t data[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x00, 0x11, 0x22, 0x33 }; // Ввод шестнадцатеричных данных

    std::cout << "Шестнадцатеричное представление::" << std::endl;
    print_in_hex(data, sizeof(data));

    std::cout << "Двоичное представление:" << std::endl;
    print_in_binary(data, sizeof(data));

    return 0;
}
```



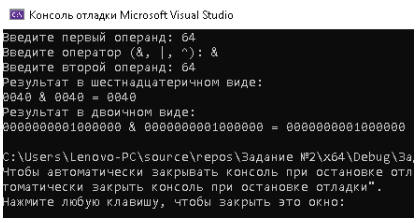


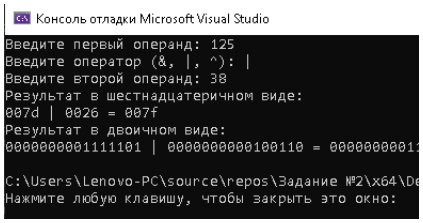
№2

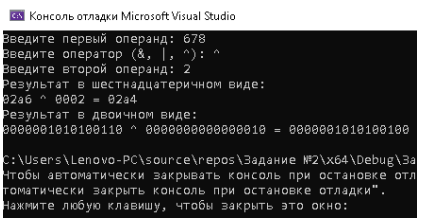
main.cpp

```cpp
#include <iostream>
#include <iomanip>
#include <bitset>
#include <sstream>

void print_in_hex(uint16_t value) { // Функция для печати uint16_t в шестнадцатеричном представлении
    std::cout << std::hex << std::setw(4) << std::setfill('0') << value;
}

void print_in_binary(uint16_t value) { // Функция для печати uint16_t в двоичном представлении
    std::cout << std::bitset<16>(value);
}

int main() {
    setlocale(LC_ALL, "RUS");

    uint16_t operand1, operand2;
    char operation;

        std::cout << "Введите первый операнд: "; // Ввод данных пользователем
    std::cin >> operand1;
    std::cout << "Введите оператор (&, |, ^): ";
    std::cin >> operation;
    std::cout << "Введите второй операнд: ";
    std::cin >> operand2;

        uint16_t result; // Выполнение операции
    switch (operation) {
    case '&':
        result = operand1 & operand2;
        break;
    case '|':
        result = operand1 | operand2;
        break;
    case '^':
        result = operand1 ^ operand2;
        break;
    default:
        std::cerr << "Неверный оператор!" << std::endl;
        return 1;
    }
```

```
        std::cout << "Результат в шестнадцатеричном виде:\n"; // Вывод результатов
    print_in_hex(operand1);
    std::cout << " " << operation << " ";
    print_in_hex(operand2);
    std::cout << " = ";
    print_in_hex(result);
    std::cout << std::endl;

    std::cout << "Результат в двоичном виде:\n";
    print_in_binary(operand1);
    std::cout << " " << operation << " ";
    print_in_binary(operand2);
    std::cout << " = ";
    print_in_binary(result);
    std::cout << std::endl;

    return 0;
}
```







№3

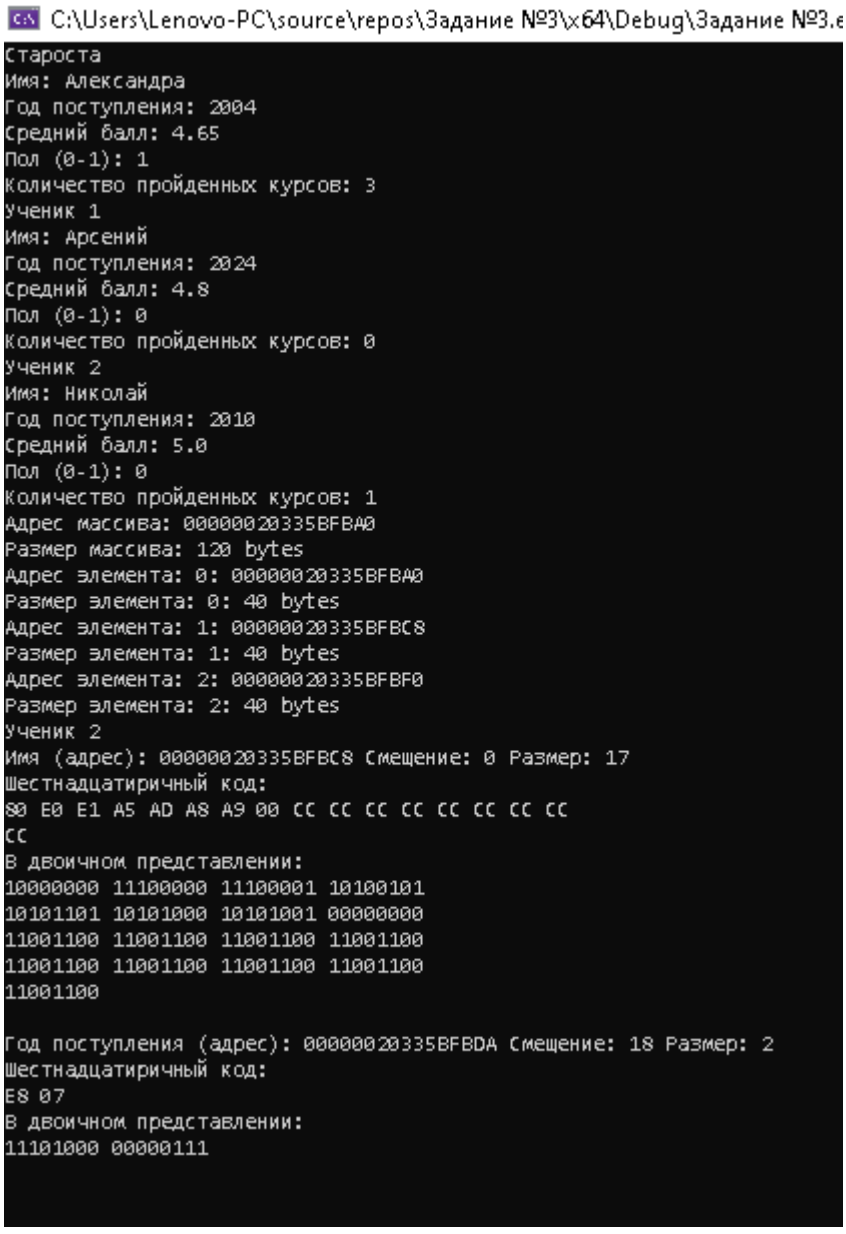
main.cpp

```
#include <iostream>
#include <cstdint>
#include <cstring> // Для использования strlen и strcpy

using namespace std;

// Структура Student, описывающая студента
struct Student {
    char Name[17];              // Имя студента (максимум 16 символов + завершающий
нулевой символ)
    uint16_t Year_of_receipt;   // Год поступления
    float Average_score;        // Средний балл
    uint16_t Sex : 1;           // Пол (0 – женский, 1 – мужской)
    uint16_t Finished_courses;  // Количество пройденных курсов
    Student* Team_lead;         // Указатель на старосту группы
};

// Вспомогательные массивы и функции для работы с шестнадцатеричным и двоичным
представлением
char digits[] = "0123456789ABCDEF";

// Функция преобразования данных в массив байтов
const uint8_t* as_bytes(const void* data) {
    return reinterpret_cast<const uint8_t*>(data);
}

// Функция преобразования nibble в шестнадцатеричный символ
char nibble_to_hex(uint8_t i) {
```

```
    return digits[i & 0xf];
}

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

// Функция вывода блока данных в шестнадцатеричном формате
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 вывода одного бита
char bit_digit(uint8_t byte, uint8_t bit) {
    return (byte & (0x1 << bit)) ? '1' : '0';
}

// Функция вывода одного байта в двоичном формате
void print_in_binary(uint8_t byte) {
    for (int bit = 7; bit >= 0; --bit) {
        cout << bit_digit(byte, bit);
    }
}

// Функция вывода блока данных в двоичном формате
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() {
    setlocale(LC_ALL, "rus");
    Student students[3];   // Массив студентов
    uint16_t gender;

    // Ввод данных для каждого студента
    for (int i = 0; i < 3; ++i) {
        if (i != 0) {
            cout << "Ученик " << i << "\n";
            students[i].Team_lead = &students[0];
        }
        else {
            students[i].Team_lead = nullptr;
```

```
            cout << "Староста\n";
        }
        cout << "Имя: ";
        cin >> students[i].Name;
        cout << "Год поступления: ";
        cin >> students[i].Year_of_receipt;
        cout << "Средний балл: ";
        cin >> students[i].Average_score;
        cout << "Пол (0-1): ";
        cin >> gender;
        students[i].Sex = gender;
        cout << "Количество пройденных курсов: ";
        cin >> students[i].Finished_courses;
    }

    // Вывод информации о массиве
    cout << "Адрес массива: " << (void*)students << "\n";
    cout << "Размер массива: " << sizeof(students) << " bytes\n";

    // Вывод информации о каждом элементе массива
    for (int i = 0; i < 3; ++i) {
        cout << "Адрес элемента: " << i << ": " << (void*)&students[i] << "\n";
        cout << "Размер элемента: " << i << ": " << sizeof(Student) << " bytes\n";
    }

    // Вывод информации о полях второго студента
    cout << "Ученик 2\n";
    cout << "Имя (адрес): " << (void*)&students[1].Name
        << " Смещение: " << offsetof(Student, Name)
        << " Размер: " << sizeof(students[1].Name) << "\nШестнадцатиричный код:\n";
    print_in_hex(&students[1].Name, sizeof(students[1].Name));
    cout << "\nВ двоичном представлении:\n";
    print_in_binary(&students[1].Name, sizeof(students[1].Name));

    cout << "\n\nГод поступления (адрес): " << (void*)&students[1].Year_of_receipt
        << " Смещение: " << offsetof(Student, Year_of_receipt)
        << " Размер: " << sizeof(students[1].Year_of_receipt) << "\nШестнадцатиричный
код:\n";
    print_in_hex(&students[1].Year_of_receipt, sizeof(students[1].Year_of_receipt));
    cout << "\nВ двоичном представлении:\n";
    print_in_binary(&students[1].Year_of_receipt, sizeof(students[1].Year_of_receipt));

    cout << "\n\nСредний балл (адрес): " << (void*)&students[1].Average_score
        << " Смещение: " << offsetof(Student, Average_score)
        << " Размер: " << sizeof(students[1].Average_score) << "\nШестнадцатиричный
код:\n";
    print_in_hex(&students[1].Average_score, sizeof(students[1].Average_score));
    cout << "\nВ двоичном представлении:\n";
    print_in_binary(&students[1].Average_score, sizeof(students[1].Average_score));

    cout << "\n\nКоличество пройденных курсов (адрес): " <<
(void*)&students[1].Finished_courses
        << " Смещение: " << offsetof(Student, Finished_courses)
        << " Размер = " << sizeof(students[1].Finished_courses) << "\nШестнадцатиричный
код:\n";
    print_in_hex(&students[1].Finished_courses, sizeof(students[1].Finished_courses));
    cout << "\nВ двоичном представлении:\n";
    print_in_binary(&students[1].Finished_courses,
sizeof(students[1].Finished_courses));

    cout << "\n\nСтароста (адрес): " << (void*)&students[1].Team_lead
```

```
        << " Смещение: " << offsetof(Student, Team_lead)
        << " Размер: " << sizeof(students[1].Team_lead) << "\nШестнадцатиричный
код:\n";
    print_in_hex(&students[1].Team_lead, sizeof(students[1].Team_lead));
    cout << "\nВ двоичном представлении:\n";
    print_in_binary(&students[1].Team_lead, sizeof(students[1].Team_lead));

    // Вывод всего массива студентов
    cout << "\nВесь массив:\n";
    print_in_hex(students, sizeof(students));
    cout << "\nВ двоичном представлении:\n";
    print_in_binary(students, sizeof(students));

    cin >> gender; // Задержка завершения программы
    return 0;
}
```

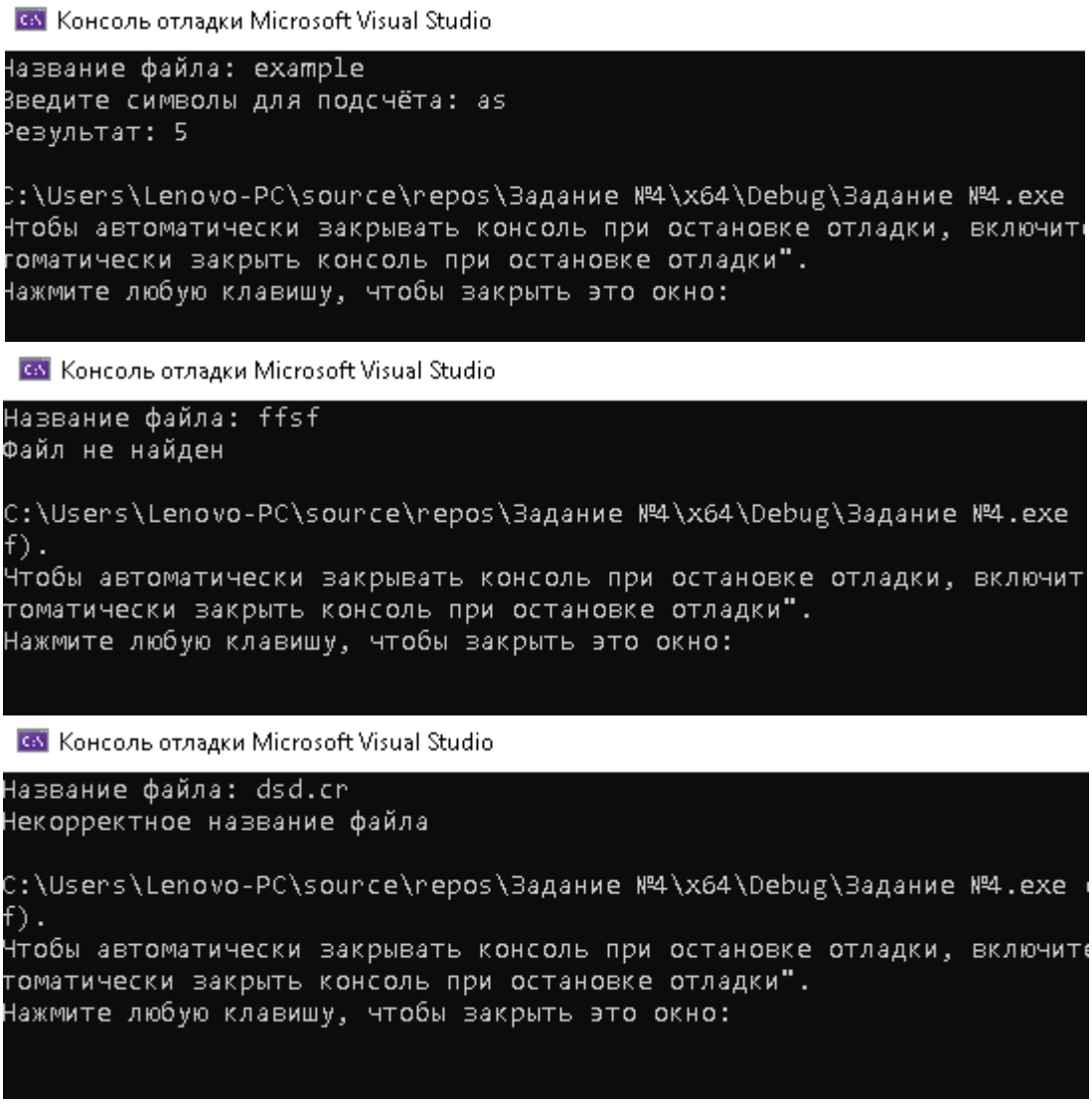


main.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cctype>
#include <locale>

using namespace std;

const size_t MAX_SIZE = 256;
const char not_allowed[] = "*\"<>?|";
const char separators[] = " \r\n,.!?:;()-";

// Функция для получения строки с пользовательским вводом
void getstring(char* s, size_t n) {
    fgets(s, n, stdin);
    if (s[strlen(s) - 1] == '\n')
        s[strlen(s) - 1] = '\0';
}

// Функция для перевода строки в нижний регистр
void lower(char* s) {
    for (; *s; s++)
        *s = tolower(*s);
}

// Функция для проверки корректности имени файла
bool iscorrectname(char* name) {
    char lext[8];

    // Проверка на наличие запрещенных символов
    for (int i = 0; name[i]; i++) {
        if (strchr(not_allowed, name[i]) != 0)
            return false;
    }

    // Проверка корректности двоеточия и пути
    char* colon = strchr(name, ':');
    if (colon != NULL) {
        if (colon - name != 1)
            return false;
        if (colon[1] != '\\')
            return false;
        if (strrchr(name, ':') != colon)
            return false;
    }

    // Проверка и добавление расширения .txt
    char* fname = strrchr(name, '\\');
    fname = fname ? fname + 1 : name;
    char* ext = strchr(fname, '.');
    if (ext == NULL)
        strcat(name, ".txt");
    else {
        strcpy(lext, ext);
        lower(lext);
        if (strcmp(lext, ".txt"))
```

```
            return false;
        }
    }
    return true;
}

int main() {
    setlocale(LC_ALL, "rus");

    char filename[MAX_SIZE];
    char str[MAX_SIZE];

    cout << "Название файла: ";
    getstring(filename, MAX_SIZE);

    // Проверка имени файла
    if (!iscorrectname(filename)) {
        cout << "Некорректное название файла\n";
        return -1;
    }

    // Открытие файла
    FILE* fin = fopen(filename, "rb");
    if (fin == NULL) {
        cout << "Файл не найден\n";
        return -1;
    }

    // Определение размера файла
    fseek(fin, 0, SEEK_END);
    size_t length = ftell(fin);
    fseek(fin, 0, SEEK_SET);

    // Чтение содержимого файла в память
    char* text = new char[length + 1];
    memset(text, 0, length + 1);
    fread(text, 1, length, fin);

    cout << "Введите символы для подсчёта: ";
    getstring(str, MAX_SIZE);

    // Подсчет вхождений строки
    const char* start = text;
    size_t count = 0;
    while (true) {
        const size_t separator_count = strspn(start, separators);
        start += separator_count;
        if (start[0] == '\0') {
            break;
        }
        const size_t word_length = strcspn(start, separators);
        if (word_length == strlen(str) && strncmp(str, start, word_length) == 0) {
            count++;
        }
        start += word_length;
    }

    cout << "Результат: " << count << "\n";

    // Освобождение памяти и закрытие файла
    fclose(fin);
    delete[] text;
```

```
    return 0;
}
```



Контрольные вопросы

1. Опишите назначение и использование операторов new и delete. Чем отличается синтаксис удаления массива от синтаксиса удаления одиночного значения?

Ответ: Операторы new и delete в языке C++ используются для динамического выделения и освобождения памяти. Они обеспечивают управление памятью в куче (heap), что позволяет создавать объекты в динамической памяти во время выполнения программы. Различие в синтаксисе:

для удаления одиночного объекта используется delete без квадратных скобок, а для удаления массива объектов используется delete[] с квадратными скобками.

2. Как при помощи динамических массивов организовать работу с квадратной матрицей, размер которой становится известен во время выполнения?

Ответ: Чтобы организовать работу с квадратной матрицей, размер которой становится известен во время выполнения, можно использовать динамические массивы. В C++ это можно сделать при помощи оператора new.

3. Что такое «рваный» массив (jagged array)? Как выделять и освобождать память под его элементы, как к ним обращаться?

Ответ: «Рваный» массив (jagged array) — это массив массивов, где каждый из подмассивов может иметь различную длину. В отличие от многомерных массивов с фиксированным размером, «рваные» массивы позволяют более гибко работать с данными, экономя память, если строки массива имеют разную длину.

4. Каковы особенности арифметики типов с плавающей запятой по сравнению с математически точной и сравнения их значений?

Ответ: Арифметика типов с плавающей запятой имеет несколько ключевых особенностей по сравнению с математически точной арифметикой:

– Ограниченная точность: Числа с плавающей запятой имеют ограниченную точность из-за ограниченного числа битов для мантиссы. Это может привести к ошибкам округления.
– Диапазон значений: Числа с плавающей запятой могут представлять как очень маленькие, так и очень большие значения, благодаря использованию экспоненты.
– Ошибки округления: Операции с числами с плавающей запятой могут вводить небольшие ошибки, поэтому прямое сравнение таких чисел часто ненадёжно. Вместо этого используется сравнение с допустимой погрешностью (эпсилон).
– Сравнение значений: Прямое сравнение чисел с плавающей запятой может быть ошибочным из-за накопления ошибок округления. В математически точной арифметике такого нет, и значения можно сравнивать напрямую.

– Оптимизация: Арифметика с плавающей запятой поддерживается аппаратно и оптимизирована для быстрого выполнения стандартных математических операций.

5. Как и почему корректно сравнивать значения с плавающей запятой и бороться с ошибками округления при операциях над такими значениями?

Ответ: Сравнение значений с плавающей запятой напрямую часто ненадежно из-за ограниченной точности представления этих чисел и ошибок округления. Чтобы корректно сравнивать такие значения и бороться с ошибками округления, используется подход с допустимой погрешностью (эпсилон).

Как и почему корректно сравнивать:

– Ошибки округления: Арифметические операции с числами с плавающей запятой могут вводить небольшие ошибки из-за ограниченной точности представления. Прямое сравнение (==) может не сработать;
– Использование эпсилон: Вместо прямого сравнения проверяется, находится ли разница между числами в пределах допустимой погрешности (эпсилон).

6. В чем заключается арифметика с фиксированной запятой, какие у нее преимущества и недостатки по сравнению с арифметикой типов с плавающей запятой?

Ответ: Арифметика с фиксированной запятой (fixed-point arithmetic) заключается в представлении чисел с использованием фиксированного количества разрядов для целой и дробной частей. Это позволяет избежать ошибок округления, характерных для арифметики с плавающей запятой, но имеет свои ограничения.

Преимущества:

– Точность: Отсутствие ошибок округления, так как числа хранятся с фиксированной точностью;
– Предсказуемость: Числа с фиксированной запятой ведут себя предсказуемо при арифметических операциях;
– Производительность: В некоторых случаях операции с фиксированной запятой могут быть быстрее, особенно на микроконтроллерах и

встраиваемых системах, где нет поддержки аппаратной арифметики с плавающей запятой.

Недостатки:

– Ограниченный диапазон: У фиксированной запятой ограниченный диапазон значений, что делает невозможным работу с очень большими или очень маленькими числами;
– Управление масштабом: Требуется явное управление масштабом (позиция запятой), что усложняет программирование;
– Гибкость: Меньшая гибкость по сравнению с плавающей запятой, так как невозможно динамически изменять позицию запятой.

7. Опишите действие оператора reinterpret_cast. В каких случаях его удобно применять, и какие проблемы при этом могут возникнуть?

Ответ: Оператор reinterpret_cast в C++ выполняет низкоуровневое приведение типа, позволяя интерпретировать битовое представление одного типа данных как другой тип данных. Он используется для преобразования указателей и интегральных типов, когда необходимо интерпретировать один тип как другой без изменения битового содержимого.

Применение reinterpret_cast:

– Преобразование указателей: Например, преобразование указателя на один тип данных в указатель на другой тип данных;
– Преобразование указателей в целые типы и обратно: Например, для работы с низкоуровневыми операциями;
– Работа с сетевыми или файловыми протоколами: Когда необходимо интерпретировать данные как другой тип для обработки или анализа.

Проблемы и риски:

– Неопределённое поведение: Приведение через reinterpret_cast может привести к неопределённому поведению, если полученный указатель используется неправильно;
– Нарушение строгой типизации: Использование reinterpret_cast может нарушать строгую типизацию C++, что затрудняет отслеживание и отладку кода;

– Потеря переносимости: Код, использующий reinterpret_cast, может быть непереносимым между различными архитектурами и компиляторами;
– Проблемы с выравниванием: Преобразование указателей на типы с различными требованиями к выравниванию может привести к сбоям.

8. Что такое выравнивание данных (alignment)? Почему оно существует, зачем и как контролировать его наличие?

Ответ: Выравнивание данных — это расположение данных в памяти с учетом определённых границ, которые зависят от архитектуры процессора. Выравнивание обеспечивает, чтобы адреса данных были кратны размеру данных или их типу (например, 2, 4, 8 байт и т.д.).

Почему выравнивание существует:

– Производительность: Современные процессоры работают быстрее, если данные выровнены по границам, соответствующим их размеру. Например, 4-байтовые данные (целые числа, float) лучше расположить по адресам, кратным 4;
– Аппаратные ограничения: Некоторые архитектуры процессоров могут обращаться к данным только по выровненным адресам;
– Простота доступа: Выровненные данные позволяют упрощать и ускорять выполнение операций чтения и записи.

Зачем контролировать выравнивание:

– Предотвращение ошибок: На некоторых архитектурах доступ к невыровненным данным может вызвать ошибки (например, ошибка сегментации);
– Оптимизация производительности: Выровненные данные обеспечивают максимальную производительность;
– Соблюдение стандартов: Определенные стандарты и соглашения требуют соблюдения правил выравнивания для корректной работы программ.

9. Как определить размер переменной в C++, и в каких случаях он отличается от размера полезных данных, связанных с переменной?

Ответ: В языке C++ для определения размера переменной используется оператор sizeof. Этот оператор возвращает размер переменной или типа данных в байтах.

Размер переменной может отличаться от размера полезных данных в следующих случаях:

– Выравнивание данных (alignment): Компилятор может добавлять отступы (padding) для выравнивания данных в структуре или классе. Это делается для оптимизации доступа к данным;
– Структуры и классы с виртуальными функциями: Наличие виртуальных функций приводит к добавлению таблицы виртуальных функций (vtable), что увеличивает размер объекта;
– Полиморфизм и наследование: Наследование может увеличивать размер объекта из-за добавления базовых классов;
– Выделение динамической памяти: При работе с указателями и выделением памяти с помощью new размер указателя отличается от фактического размера выделенных данных.

10. Какие побитовые операторы имеются в C++? Приведите примеры их работы.

Ответ: В языке C++ доступны несколько побитовых операторов, которые позволяют выполнять операции над отдельными битами целочисленных данных. Основные из них:

– Побитовое И (AND) — &. Каждый бит результата равен 1, если оба соответствующих бита операндов равны 1;
– Побитовое ИЛИ (OR) — |. Каждый бит результата равен 1, если хотя бы один из соответствующих битов операндов равен 1;
– Побитовое исключающее ИЛИ (XOR) — ^. Каждый бит результата равен 1, если соответствующие биты операндов различны;
– Побитовое НЕ (NOT) — ~. Инвертирует каждый бит операнда (1 становится 0, и 0 становится 1);
– Побитовый сдвиг влево — <<. Сдвигает биты операнда влево на указанное количество позиций;
– Побитовый сдвиг вправо — >>. Сдвигает биты операнда вправо на указанное количество позиций.

11. Что такое битовые флаги и битовые маски? Как в C++ записываются числа в системах счисления, отличных от десятичной?

Ответ: Битовые флаги и битовые маски используются для управления и проверки значений отдельных битов в целочисленных данных. Битовый флаг — это отдельный бит в числе, который используется для хранения булевого значения (0 или 1). Битовые флаги позволяют компактно представлять состояния и атрибуты. Например, можно использовать отдельные биты для хранения нескольких флагов в одной переменной. Битовая маска — это значение, используемое для установки, очистки или проверки значений отдельных битов с помощью битовых операций. Запись осуществляется следующим образом: Десятичная в десятичном формате, шестнадцатиричная с префиксом 0x, восьмеричная с префиксом 0, двоичная с префиксом 0b.

12. Каким образом можно: а) объявить целочисленную переменную размером 8, 16, 32 бита (гарантированно); б) объявить битовый массив произвольно большого размера; в) поле структуры размера, не кратного байту (например, 9 бит)?

Ответ:

а) Объявление целочисленных переменных фиксированного размера (8, 16, 32 бита). Для этого используются типы из заголовочного файла <cstdint>, такие как std::int8_t, std::int16_t, std::int32_t, а также их беззнаковые аналоги (std::uint8_t, std::uint16_t, std::uint32_t);

б) Объявление битового массива произвольно большого размера. Для создания битового массива можно использовать std::vector<bool> или битовые операции над массивом целых чисел;

в) Поле структуры размера, не кратного байту (например, 9 бит). Для этого используются битовые поля в структурах. Например: std::uint16_t field1 : 9.

13. Как в C++ объявляются простые массивы (одно- и многомерные), каков синтаксис их инициализации, доступа к отдельным элементам, определения размера?

Ответ: Одномерный массив — это последовательность элементов одного типа, расположенных в памяти подряд. Синтаксис объявления одномерного массива в C++: тип данных имя одномерного массива [размерность одномерного массива]. Пример: int a . Синтаксис инициализации одномерного массива: элементы инициализируются конкретными значениями при

объявлении путём заключения этих значений в фигурные скобки {}. Пример: int a = { 5, -12, -12, 9, 10, 0, -9, -12, -1, 23, 65, 64, 11, 43, 39, -15 }; Синтаксис доступа к отдельным элементам массива: к значению любого элемента массива можно получить доступ так же, как и к значению обычной переменной того же типа. Синтаксис: name[index]. Синтаксис определения размера массива: для определения размера массива в байтах можно использовать оператор sizeof. Он возвращает полный размер массива в байтах, то есть размер элемента, умноженный на размер массива.

14. Опишите шаблон класса std::array<T, N>, его назначение и преимущества использования по сравнению с простыми массивами.

Ответ: Шаблон класса std::array<T, N> — это контейнер для фиксированных размеров массивов, предоставляемый в стандартной библиотеке C++ (начиная с C++11). Этот класс объединяет удобство использования массивов с безопасностью и функциональностью стандартных контейнеров STL.

Назначение:

– std::array<T, N> предоставляет массив фиксированного размера N для элементов типа T. Он наследует многие преимущества обычных массивов, но также добавляет функциональность и безопасность, свойственные стандартным контейнерам STL.

Преимущества:

– Типизированная безопасность и контейнерные методы: std::array предоставляет методы, типичные для контейнеров STL (begin, end, size, fill и т. д.), что упрощает работу с массивами и позволяет использовать их в алгоритмах STL;

– Размер компилируется временем: Размер массива является частью его типа и известен на этапе компиляции, что позволяет компилятору выполнять оптимизации и предотвращать ошибки, связанные с неправильным использованием размеров массива;

– Семантика копирования: std::array поддерживает семантику копирования и присваивания, что делает его более удобным для использования в функциях и возвращаемых значениях;

– Строгие гарантии безопасности: Методы std::array предоставляют безопасный доступ к элементам массива и предотвращают ошибки, связанные с выходом за границы массива (например, метод at);

– Совместимость с алгоритмами STL: std::array легко интегрируется с алгоритмами STL, что делает его более мощным и удобным для использования в различных задачах.

15. Что такое строки в стиле C (C-style string) и какие средства работы с ними имеются в стандартной библиотеке C++?

Ответ: Строки в стиле C (C-style strings) представляют собой массивы символов, оканчивающихся нулевым символом (\0). Нулевой символ используется для обозначения конца строки. Такие строки широко используются в языке программирования C и поддерживаются в C++. Основные средства из стандартной библиотеки: strlen, strcpy, strcat, strcmp, strncpy, strchr, strstr.