

Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 1
по дисциплине «Системное программирование»
УКАЗАТЕЛИ, СТРУКТУРЫ И ОБЪЕДИНЕНИЯ

Студент гр. БИБ 211

_____ В.Ф. Санников

«___» _____ 2023 г.

Руководитель

Старший преподаватель

_____ В.И. Морозов

«___» _____ 2023 г.

Москва

СОДЕРЖАНИЕ

1 Задание на практическую работу.....	3
2 Ход работы.....	5
2.1 Программа сериализации и зашифрования входных данных.....	6
2.2 Программа дешифрования и десериализации данных.....	11
2.3 Результаты выполнения программ.....	14
3 Выводы о проделанной работе.....	16
ПРИЛОЖЕНИЕ А.....	16
ПРИЛОЖЕНИЕ Б.....	19

1 Задание на практическую работу

Целью работы является изучение явления сериализации и десериализации в низкопрофильных языках программирования, а также написание программ на языке С, которые соответствуют поставленным условиям.

В рамках практической работы необходимо выполнить следующее:

- 1) Повторно ознакомиться с конспектами семинара 1 и практического занятия 1, материалами лекций, а также рассмотреть теоретическую часть из данного файла (раздел 1).
- 2) С помощью языка программирования Си (не С++) необходимо написать две программы, одна из которых выполняет сериализацию и простое зашифрование введенных пользователем данных, а также запись результата в файл; другая – расшифрование данных из файла, десериализацию этих данных и вывод пользователю в консоль. При этом:

а) Программа должна содержать, как минимум, одну структуру (struct) и одно объединение (union).

б) Структура должна содержать поля, хранящие данные, вводимые пользователем (по варианту).

в) Объединение должно включать в качестве элементов объект структуры из подпункта б данного списка и массив байт, по размеру равный размеру объекта этой структуры.

г) Преобразование данных, полученных от пользователя, в байты должно производиться с помощью структуры и объединения.

д) Программа должна содержать, как минимум, две функции, одна из которых принимает на вход байт, модифицирует его и возвращает значение модифицированного байта, а вторая принимает на вход указатели на два буфера (массива байт) – входной и выходной –, их размер, а также функцию обратного вызова и записывать результат применения функции к элементам входного буфера на соответствующие позиции выходного буфера.

е) Простое шифрование данных, полученных от пользователя, должно производиться с помощью двух функций, описанных в подпункте д этого списка.

ж) Программа должна сохранять значение зашифрованных данных в массив байт, выделенный в динамической памяти (на куче).

3) Для выполнения задания может использоваться любая среда разработки. При выполнении необходимо учитывать следующие требования:

а) Задача считается решённой, если необходимое по варианту действие выполняется для всех допустимых входных данных.

б) Для работы с файлами и консолью должны использоваться только специфичные для языка Си функции из заголовочного файла `stdio.h`. Использование аналогов из языка C++ не допускается. Например, вывод в консоль следует делать с помощью функции `printf`.

в) Программа должна корректно завершаться, не вызывая аварийный останов.

г) Программа должна брать название файла из аргумента, переданного при запуске в консоли. В случае, если количество переданных аргументов не равно ожидаемому, программа должна вывести подсказку для пользователя, поясняющую правила её (программы) использования.

д) Программа должна запрашивать данные для сериализации у пользователя после запуска. Для этого должно быть выведено приглашение ко вводу, описывающее, какие данные пользователю необходимо ввести в данный момент.

е) Возвращаемые значения всех вызываемых функций должны проверяться на предмет возникновения ошибок. В случае возникновения ошибки необходимо вывести сообщение, оповещающее пользователя о произошедшем, содержащее в обязательном порядке код ошибки и её текстовое описание. В случае, если в результате возникшей ошибки программа должна быть завершена, перед завершением необходимо освободить все занятые ресурсы (очистить выделенную память, закрыть открытые файлы).

ж) Допускается ограничить размер буфера входных данных, вводимых пользователем с консоли. В таком случае приглашение ко вводу должно содержать информацию об этом ограничении.

4) Оформить отчёт в соответствии с требованиями из файла «Требования к оформлению отчёта по практической работе». Не забыть прикрепить к отчёту исходные коды обеих программ в виде приложения. В отчёте необходимо отразить, как минимум:

а) Результаты работы программы в виде снимков экрана, демонстрирующих работу программы для 2-3 вариантов верных и для каждого типа неверных входных данных (чтобы продемонстрировать все предусмотренные сообщения об ошибках).

б) Процесс сборки и запуска программы.

в) Принципы работы алгоритмов, реализованных в коде и саму реализацию этих алгоритмов в виде листингов кода.

г) Изменения ключевых значений (в переменных) в ходе работы программы.

2 Ход работы

В ходе работы мне было необходимо написать две программы на языке C, одна из которых производит сериализацию данных и последующее шифрование, а вторая программа расшифровывает эти данные и десериализует их.

Сериализация - это процесс преобразования данных в структурированный формат (например, в байтовый поток), который можно сохранить или передать.

Десериализация - это обратный процесс, при котором данные из сериализованного формата преобразуются обратно в их исходное представление.

Шифрование и расшифрование данных в рамках данного задания заключается в несложных алгоритмах, которые оперируют над байтами данных. Сложность заключается в правильной реализации шифрования и последующего правильного расшифрования данных. Сборка программы реализована с помощью компилятора gcc. Запуск приложения реализован через команду cmd.

2.1 Программа сериализации и зашифрования входных данных

Взглянув на список моей группы, можно заметить, что я занимаю в ней 19 место из 27, а значит мне необходимо выполнить работу с входными данными и алгоритмом шифрования и расшифрования из варианта номер 4.

Начать написание программы стоит с подключения необходимых библиотек:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

Листинг 1 – подключение библиотек.

`<stdio.h>` - эта библиотека содержит функции для ввода/вывода данных, такие как `printf()` для вывода данных на экран и `scanf()` для ввода данных с клавиатуры. Она также содержит функции для работы с файлами, такие как `fopen()` для открытия файла и `fclose()` для закрытия файла.

`<stdlib.h>` - эта библиотека содержит функции для работы с динамической памятью, такие как `malloc()` для выделения памяти и `free()` для освобождения памяти. Она также содержит функции для работы с числами, такие как `rand()` для генерации случайных чисел.

`<string.h>` - эта библиотека содержит функции для работы со строками, такие как `strcpy()` для копирования строк, `strcat()` для объединения строк и `strlen()` для определения длины строки.

`<ctype.h>` - эта библиотека содержит функции для работы с символами, такие как `isdigit()` для проверки, является ли символ цифрой, `isalpha()` для проверки, является ли символ буквой и т. д.

Далее нам необходимо определить структуры и объединения для последующего использования в программе:

```
struct UserData {
    char name[50];
    char surname[50];
    char patronymic[50];
    int identifier;
};
```

```
union SerializedData {
    struct UserData user;
    unsigned char bytes[sizeof(struct UserData)];
};
```

Листинг 2 – объявление структуры данных и объединения union

В этой части кода происходит определение структуры UserData, которая содержит поля для имени, фамилии, отчества и идентификатора, а также определение объединения (union) SerializedData, которое содержит структуру UserData и массив байтов того же размера.

В языке C "структура" и "юнион" - это специальные типы данных, которые позволяют объединять разные типы данных в одном. Разница между ними заключается в том, как они хранят данные. Структура - это тип данных, который позволяет комбинировать различные типы данных под одним именем. Поля - это переменные внутри структуры, каждая собственной уникальной именем и типом данных. Они определяются внутри фигурных скобок {} при объявлении структуры. Структуры часто используются для описания сложных данных, таких как описание человека с именем, возрастом и адресом.

Юнион - это тип данных, который позволяет хранить несколько разных типов данных в одной и той же области памяти. Юнион имеет несколько полей, но занимает память только для самого большого поля. Поля разделяют одну и ту же область памяти.

Юнионы часто используются для экономии памяти или для обработки разных типов данных через общую область памяти.

В обоих случаях, структуры и юнионы позволяют объединять различные типы данных вместе под одним именем, но с разными правилами хранения и доступа.

Следующим шагом написания программы нам необходимо определить функцию modifyByte, которая выполняет простую операцию XOR над байтом, определить функцию applyCallback, которая применяет заданную функцию обратного вызова к каждому байту входного буфера и сохраняет результат в выходном буфере, и наконец определить функцию simpleEncryption, которая использует функцию applyCallback для применения простой операции шифрования к входному буферу.

```

unsigned char modifyByte(unsigned char byte) {
    return byte ^ 0b01010101;
}

void applyCallback(unsigned char* inputBuffer, unsigned char* outputBuffer, int size,
unsigned char (*callback)(unsigned char)) {
    for (int i = 0; i < size; i++) {
        outputBuffer[i] = callback(inputBuffer[i]);
    }
}

void simpleEncryption(unsigned char* inputBuffer, unsigned char* outputBuffer, int size) {
    applyCallback(inputBuffer, outputBuffer, size, modifyByte);
}

```

Листинг 3 – определение функций кодирования и сохранения данных.

Функция `modifyByte` принимает на вход байт (один байт данных) и ключ (также один байт) и выполняет операцию XOR над этими двумя значениями. Операция XOR возвращает 1, если биты в операндах различны, и 0, если они одинаковы. Таким образом, функция `modifyByte` выполняет простую операцию шифрования/дешифрования данных.

Функция `applyCallback` принимает на вход входной буфер, выходной буфер, длину буфера и функцию обратного вызова. Она применяет заданную функцию обратного вызова к каждому байту входного буфера и сохраняет результат в выходном буфере.

Функция `simpleEncryption` использует функцию `applyCallback` для применения простой операции шифрования к входному буферу. То есть, она применяет функцию `modifyByte` к каждому байту входного буфера с использованием заданного ключа и сохраняет результат в выходном буфере. Это позволяет зашифровать данные во входном буфере с использованием простой операции XOR.

Следующим шагом помимо проверки данных необходимо реализовать функцию мэйн, которая будет принимать пользовательский ввод и обрабатывать его с использованием описанных ранее функций:

```

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <output_filename>\n", argv[0]);
        return 1;
    }
}

```



```

char outputFilename[100];
strcpy(outputFilename, argv[1]);

struct UserData userData;
char input[150]; // Буфер для ввода пользователя

printf("Maximum 50 characters for first name, last name, and middle name.\n");

printf("Enter last name: ");
fgets(input, sizeof(input), stdin);
input[strcspn(input, "\n")] = 0; // Удаляем символ новой строки

if (!isAlphaWithSpace(input)) {
    printf("Error: Invalid input for last name!\n");
    return 1;
}

strncpy(userData.name, input, sizeof(userData.name));

printf("Enter first name: ");

fgets(input, sizeof(input), stdin);
input[strcspn(input, "\n")] = 0; // Удаляем символ новой строки
if (!isAlphaWithSpace(input)) {
    printf("Error: Invalid input for first name!\n");
    return 1;
}

strncpy(userData.surname, input, sizeof(userData.surname));

printf("Enter middle name: ");

fgets(input, sizeof(input), stdin);
input[strcspn(input, "\n")] = 0; // Удаляем символ новой строки
if (!isAlphaWithSpace(input)) {
    printf("Error: Invalid input for middle name!\n");
    return 1;
}

strncpy(userData.patronymic, input, sizeof(userData.patronymic));

printf("Maximum 10 characters for ID.\n");

```

```

char idInput[20];
int validInput = 0;
while (!validInput) {
    printf("Enter ID: ");
    fgets(idInput, sizeof(idInput), stdin);
    idInput[strcspn(idInput, "\n")] = 0; // Удаление символа новой строки
    int i;
    for (i = 0; idInput[i] != '\0'; i++) {
        if (!isdigit(idInput[i])) {
            printf("Error: ID must be a number!\n");
            break;
        }
    }
    if (idInput[i] == '\0') {
        validInput = 1;
        userData.identifier = atoi(idInput);
    }
}

union SerializedData serializedData;
serializedData.user = userData;

unsigned char *encryptedBytes = (unsigned char*)malloc(sizeof(struct UserData));

simpleEncryption(serializedData.bytes, encryptedBytes, sizeof(struct UserData));

FILE *file = fopen(outputFilename, "wb");
if (file != NULL) {
    fwrite(encryptedBytes, sizeof(unsigned char), sizeof(struct UserData), file);
    fclose(file);
} else {
    fprintf(stderr, "Error! File does not exist!\n");
    return 1;
}

free(encryptedBytes);

return 0;
}

```

Листинг 4 – основная функция программы.

Функция `main` представляет собой точку входа в программу. Она принимает аргументы командной строки `argc` и `argv`, где `argc` - количество аргументов командной строки, а `argv` - массив строк, содержащий сами аргументы.

Сначала функция `main` проверяет, что был передан один аргумент командной строки - имя файла, в который будет записан зашифрованный результат. Если количество аргументов не равно 2, программа выводит сообщение об использовании и завершает свою работу с кодом ошибки 1. Затем создается экземпляр структуры `UserData`, которая содержит поля для имени, фамилии, отчества и идентификатора пользователя. Также создается буфер `input` для ввода данных пользователя.

После этого программа запрашивает у пользователя ввод фамилии, проверяет его на наличие только букв и пробелов, копирует введенное значение в поле структуры `userData.name` и также делает для полей `userData.surname` и `userData.patronymic`.

Далее программа запрашивает ввод идентификатора пользователя, проверяет его на наличие только цифр, и сохраняет его в поле `userData.identifier`. Затем создается экземпляр `union SerializedData`, который содержит структуру `UserData` и массив байтов, представляющих эту структуру.

Далее выделяется память для массива `encryptedBytes`, который будет содержать зашифрованные данные. Функция `simpleEncryption` вызывается для зашифровки данных из `serializedData.bytes` в массив `encryptedBytes`. Затем программа открывает файл с именем `outputFilename` для записи в бинарном режиме. Если файл открыт успешно, зашифрованные данные записываются в файл с помощью функции `fwrite`, после чего файл закрывается. Если файл не открыт, программа выводит сообщение об ошибке и завершает свою работу с кодом ошибки 1. Наконец, освобождается выделенная память для массива `encryptedBytes`, и функция `main` завершает свою работу с кодом успешного завершения 0.

Таким образом, функция `main` выполняет ввод данных пользователя, их проверку на корректность, шифрование этих данных и запись зашифрованных данных в файл.

С полным кодом программы сериализации можно ознакомиться в ПРИЛОЖЕНИИ А.

2.2 Программа дешифрования и десериализации данных

Данные, полученные в процессе выполнения программы сериализации и шифрования данных в конце выполнения программы были сохранены в файле с расширением `.bin`.

Программа про которую сейчас пойдёт речь, дешифрует данные из этого файла и производит десериализацию и последующий вывод данных, которые изначально подавались на вход.

Первым делом как и в предыдущем коде мы вызываем необходимые библиотеки:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Листинг 5 – подключение библиотек.

Следующим шагом точно так же как и в предыдущей кодовой реализации, мы задаём структуры данных, поля и объединения. После этого определяются функции, выполняющие обратные действия к функциям из первого кода.

```
struct UserData { // Обновленная структура
    char name[50];
    char surname[50];
    char patronymic[50];
    int identifier;
};

union SerializedData {
    struct UserData user; // Используем ту же структуру, что и в первой программе
    unsigned char bytes[sizeof(struct UserData)];
};

unsigned char modifyByte(unsigned char byte) {
    return byte ^ 0b01010101; // Модификация байта (для дешифрования)
}

void applyCallback(unsigned char* inputBuffer, unsigned char* outputBuffer, int size,
    unsigned char (*callback)(unsigned char)) {
    for (int i = 0; i < size; i++) {
        outputBuffer[i] = callback(inputBuffer[i]);
    }
}

void simpleDecryption(unsigned char* inputBuffer, unsigned char* outputBuffer, int size) {
    applyCallback(inputBuffer, outputBuffer, size, modifyByte);
}
```

Листинг 6 – определения структур, полей и объединений. Определение функций.

Функция `main` в данном коде представляет собой точку входа в программу. Она принимает аргументы командной строки `argc` и `argv`, где `argc` - количество аргументов командной строки, а `argv` - массив строк, содержащий сами аргументы.

Сначала функция `main` проверяет, что был передан один аргумент командной строки - имя файла, из которого будет считан зашифрованный результат. Если количество аргументов не равно 2, программа выводит сообщение об использовании и завершает свою работу с кодом ошибки 1.

Затем программа открывает файл с именем `inputFilename` для чтения в бинарном режиме. Если файл открыт успешно, программа определяет его размер, выделяет память для массива `encryptedBytes` и считывает зашифрованные данные из файла в этот массив. После этого файл закрывается. Если файл не открыт, программа выводит сообщение об ошибке и завершает свою работу с кодом ошибки 1.

Затем выделяется память для массива `decryptedBytes`, который будет содержать дешифрованные данные. Функция `simpleDecryption` вызывается для дешифрования данных из массива `encryptedBytes` в массив `decryptedBytes`.

Затем создается экземпляр union `SerializedData`, который содержит структуру `UserData` и массив байтов, представляющих эту структуру. Далее данные из массива `decryptedBytes` копируются в поле `serializedData.bytes` структуры `SerializedData`. Наконец, данные из структуры `serializedData` выводятся на экран, и освобождается выделенная память для массивов `encryptedBytes` и `decryptedBytes`. Функция `main` завершает свою работу с кодом успешного завершения 0.

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <input_filename>\n", argv[0]);
        return 1;
    }

    char inputFilename[100];
    strcpy(inputFilename, argv[1]);

    FILE *file = fopen(inputFilename, "rb");
    if (file != NULL) {
        fseek(file, 0, SEEK_END);
        int fileSize = ftell(file);
        rewind(file);

        unsigned char *encryptedBytes = (unsigned char*)malloc(fileSize);
        fread(encryptedBytes, sizeof(unsigned char), fileSize, file);
        fclose(file);

        unsigned char *decryptedBytes = (unsigned char*)malloc(sizeof(struct UserData)); //
Обновленная структура
        simpleDecryption(encryptedBytes, decryptedBytes, sizeof(struct UserData)); //
Используем обновленную структуру

        union SerializedData serializedData;
        serializedData.user = *(struct UserData*)decryptedBytes; // Используем обновленную
структуру

        printf("Last name: %s\n", serializedData.user.name);
        printf("First name: %s\n", serializedData.user.surname);
        printf("Middle name: %s\n", serializedData.user.patronymic);
        printf("ID: %d\n", serializedData.user.identifier);

        free(encryptedBytes);
    }
```

```

        free(decryptedBytes);
    } else {
        fprintf(stderr, "Failed to open the file for reading\n");
    }

    return 0;
}

```

Листинг 7 - основная функция программы.

С полным кодом программы можно ознакомиться в ПРИЛОЖЕНИИ Б.

2.3. Результаты выполнения программ

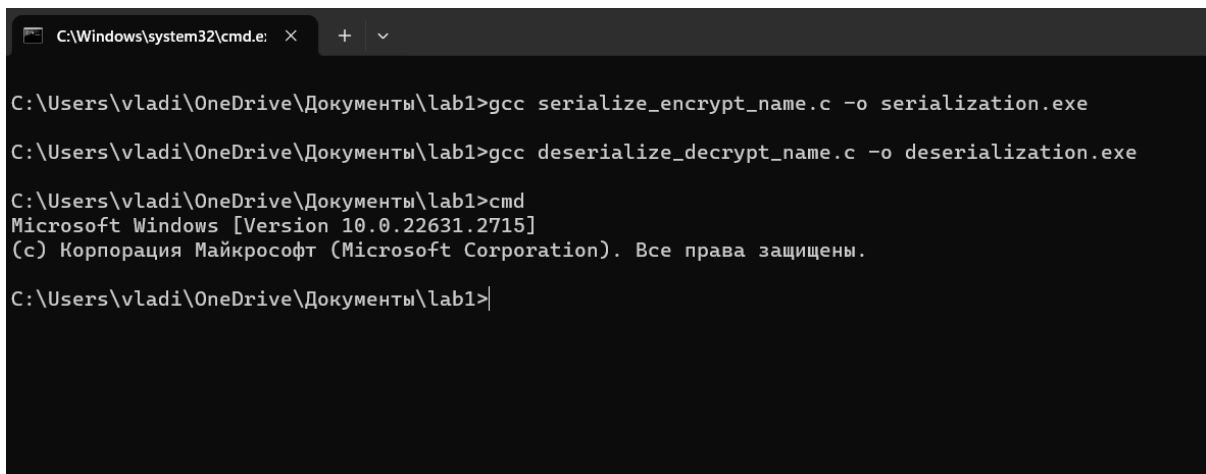
Чтобы оптимизировать процесс сборки двух программ, напишем просто .bat файл:
gcc serialize_encrypt_name.c -o serialization.exe

gcc deserialize_decrypt_name.c -o deserialization.exe

cmd

Листинг 8 – бат-файл для оптимизации сборки программ с помощью gcc.

В результате запуска бат-файла мы можем видеть следующий вывод:



```

C:\Users\vladi\OneDrive\Документы\lab1>gcc serialize_encrypt_name.c -o serialization.exe
C:\Users\vladi\OneDrive\Документы\lab1>gcc deserialize_decrypt_name.c -o deserialization.exe
C:\Users\vladi\OneDrive\Документы\lab1>cmd
Microsoft Windows [Version 10.0.22631.2715]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\vladi\OneDrive\Документы\lab1>

```

Картинка 1 – результат запуска бат-файла.

Можно заметить, что при сборке не было обнаружено никаких ошибок в написанном коде и теперь мы можем запустить первую программу в этом же окне с помощью команды:

serialization.exe data.bin

Листинг 9 – команда для запуска первой программы для сериализации и шифрования.

После того, как программа запустилась, она предлагает нам ввести входные данные, указывая при этом, в каком виде они должны подаваться.

```
C:\Users\vladi\OneDrive\Документы\lab1>serialization.exe data.bin
Maximum 50 characters for first name, last name, and middle name.
Enter last name: Sannikov
Enter first name: Vladimir
Enter middle name: Alexeevich
Maximum 10 characters for ID.
Enter ID: 4
```

Картинка 2 – ввод данных в первую программу и её выполнение.

Если произойдёт какая-то ошибка, программа нас о ней уведомит. Выглядит это примерно так:

```
C:\Users\vladi\OneDrive\Документы\lab1>serialization.exe data.bin
Maximum 50 characters for first name, last name, and middle name.
Enter last name: 345
Error: Invalid input for last name!
```

Картинка 3 – вывод ошибки входных данных.

Если же никаких ошибок не вывелось, значит вероятнее всего всё прошло успешно и сериализованные и зашифрованные данные сохранились в указанных файл data.bin. Теперь в этом же окне командной строки мы можем запустить вторую программу с помощью команды:

serialization.exe data.bin

Листинг 10 – команда для запуска программы десериализации.

В результате если входные данные были введены пользователем корректно, то и вывод будет именно такой, каким мы ожидаем его увидеть, то есть он будет идентичен вводу пользователя. Это будет означать что программа верно произвела дешифрование и десериализацию данных из последовательности байт информации, которая была записана в файле data.bin.

```
C:\Users\vladi\OneDrive\Документы\lab1>deserialization.exe data.bin
Last name: Sannikov
First name: Vladimir
Middle name: Alexeevich
ID: 4
```

Картинка 4 – вывод программы десериализации.

Как можно заметить, программа отработала корректно. Стоит заметить, что если на вход программы сериализации подать больше символов, чем программа просит, то программа завершит свою работу и уведомит пользователя об ошибке. Если ввести больше букв ФИО, чем 50 в каждом из предложенных, буфер переполнится и программа завершит работу.. В случае идентификатора обладателя ФИО, при

переполнении типа `int`, мы также получим некорректные выходные данные. При этом стоит отметить, что программа работает только с латиницей и арабскими цифрами.

3 Выводы о проделанной работе

В результате выполнения данной лабораторной работы мне удалось попробовать себя в написании кода на низкоуровневом языке программирования `C`, а также успешно реализовать поставленную задачу. Я написал два кода, как и требовалось в задании, которые правильно работают а также выводят уведомления об ошибках, если они есть. Поскольку я никогда в своей жизни не писал на языке `C` подобный код, задача оказалось очень трудно реализуемой для меня, но не невозможной. Я смог не просто изучить какие-то понятия, но и попробовать поработать с такими, как поля, объединения, кучи (динамическая память). В целом считаю результат выполнения работы успешным.

ПРИЛОЖЕНИЕ А

Листинг полного кода сериализации и кодирования.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct UserData {
    char name[50];
    char surname[50];
    char patronymic[50];
    int identifier;
};

union SerializedData {
    struct UserData user;
    unsigned char bytes[sizeof(struct UserData)];
};

unsigned char modifyByte(unsigned char byte) {
    return byte ^ 0b01010101;
}
```



```

void applyCallback(unsigned char* inputBuffer, unsigned char* outputBuffer, int size,
unsigned char (*callback)(unsigned char)) {
    for (int i = 0; i < size; i++) {
        outputBuffer[i] = callback(inputBuffer[i]);
    }
}

void simpleEncryption(unsigned char* inputBuffer, unsigned char* outputBuffer, int size) {
    applyCallback(inputBuffer, outputBuffer, size, modifyByte);
}

int isAlphaWithSpace(const char *str) {
    while (*str) {
        if (!isalpha(*str) && !isspace(*str)) {
            return 0; // Не только буквы и пробелы
        }
        str++;
    }
    return 1; // Все символы - буквы и пробелы
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <output_filename>\n", argv[0]);
        return 1;
    }

    char outputFilename[100];
    strcpy(outputFilename, argv[1]);

    struct UserData userData;
    char input[150]; // Буфер для ввода пользователя

    printf("Maximum 50 characters for first name, last name, and middle name.\n");

    printf("Enter last name: ");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = 0; // Удаляем символ новой строки

    if (!isAlphaWithSpace(input)) {
        printf("Error: Invalid input for last name!\n");
        return 1;
    }
}

```

```

strcpy(userData.name, input, sizeof(userData.name));

printf("Enter first name: ");

fgets(input, sizeof(input), stdin);
input[strcspn(input, "\n")] = 0; // Удаляем символ новой строки
if (!isAlphaWithSpace(input)) {
    printf("Error: Invalid input for first name!\n");
    return 1;
}

strcpy(userData.surname, input, sizeof(userData.surname));

printf("Enter middle name: ");

fgets(input, sizeof(input), stdin);
input[strcspn(input, "\n")] = 0; // Удаляем символ новой строки
if (!isAlphaWithSpace(input)) {
    printf("Error: Invalid input for middle name!\n");
    return 1;
}

strcpy(userData.patronymic, input, sizeof(userData.patronymic));

printf("Maximum 10 characters for ID.\n");

char idInput[20];
int validInput = 0;
while (!validInput) {
    printf("Enter ID: ");
    fgets(idInput, sizeof(idInput), stdin);
    idInput[strcspn(idInput, "\n")] = 0; // Удаление символа новой строки
    int i;
    for (i = 0; idInput[i] != '\0'; i++) {
        if (!isdigit(idInput[i])) {
            printf("Error: ID must be a number!\n");
            break;
        }
    }
    if (idInput[i] == '\0') {
        validInput = 1;
        userData.identifier = atoi(idInput);
    }
}

```

```

union SerializedData serializedData;
serializedData.user = userData;

unsigned char *encryptedBytes = (unsigned char*)malloc(sizeof(struct UserData));

simpleEncryption(serializedData.bytes, encryptedBytes, sizeof(struct UserData));

FILE *file = fopen(outputFilename, "wb");
if (file != NULL) {
    fwrite(encryptedBytes, sizeof(unsigned char), sizeof(struct UserData), file);
    fclose(file);
} else {
    fprintf(stderr, "Error! File does not exist!\n");
    return 1;
}

free(encryptedBytes);

return 0;
}

```

ПРИЛОЖЕНИЕ Б

Листинг полного кода десериализации и расшифровки.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct UserData { // Обновленная структура
    char name[50];
    char surname[50];
    char patronymic[50];
    int identifier;
};

union SerializedData {
    struct UserData user; // Используем ту же структуру, что и в первой программе
    unsigned char bytes[sizeof(struct UserData)];
};

```

```

unsigned char modifyByte(unsigned char byte) {
    return byte ^ 0b01010101; // Модификация байта (для дешифрования)
}

void applyCallback(unsigned char* inputBuffer, unsigned char* outputBuffer, int size,
unsigned char (*callback)(unsigned char)) {
    for (int i = 0; i < size; i++) {
        outputBuffer[i] = callback(inputBuffer[i]);
    }
}

void simpleDecryption(unsigned char* inputBuffer, unsigned char* outputBuffer, int size) {
    applyCallback(inputBuffer, outputBuffer, size, modifyByte);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <input_filename>\n", argv[0]);
        return 1;
    }

    char inputFilename[100];
    strcpy(inputFilename, argv[1]);

    FILE *file = fopen(inputFilename, "rb");
    if (file != NULL) {
        fseek(file, 0, SEEK_END);
        int fileSize = ftell(file);
        rewind(file);

        unsigned char *encryptedBytes = (unsigned char*)malloc(fileSize);
        fread(encryptedBytes, sizeof(unsigned char), fileSize, file);
        fclose(file);

        unsigned char *decryptedBytes = (unsigned char*)malloc(sizeof(struct UserData)); //
Обновленная структура
        simpleDecryption(encryptedBytes, decryptedBytes, sizeof(struct UserData)); //
Используем обновленную структуру

        union SerializedData serializedData;
        serializedData.user = (struct UserData*)decryptedBytes; // Используем обновленную
структуру

```

```
printf("Last name: %s\n", serializedData.user.name);
printf("First name: %s\n", serializedData.user.surname);
printf("Middle name: %s\n", serializedData.user.patronymic);
printf("ID: %d\n", serializedData.user.identifier);

free(encryptedBytes);
free(decryptedBytes);
} else {
    fprintf(stderr, "Failed to open the file for reading\n");
}

return 0;
}
```