Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Конструирование программ»

ОТЧЕТ

к лабораторной работе №8

на тему:

«ИСПОЛЬЗОВАНИЕ АССЕМБЛЕРНЫХ ПРЕРЫВАНИЙ В C++»

БГУИР 1-40-04-01

Выполнил студент группы 253501
Малюш Денис Олегович

_____
(дата, подпись студента)

Проверил старший преподаватель
Ячин Николай Сергеевич

_____
(дата, подпись преподавателя)

Минск 2023

Задание: Менеджер паролей.

На стороне Assembler:

Реализация алгоритма поточного шифрования RC4, используя assembler-функции.

Генератор случайных чисел: На базе системного таймера или других источников энтропии для создания случайных паролей.

Пользователь может сохранять пароли и другую конфиденциальную информацию. Данные хранятся в зашифрованном виде. Реализуйте функцию генерации случайных паролей.

Тайм-аут бездействия: Если менеджер паролей открыт и не используется в течение заданного времени, автоматически блокируйте его.

На стороне C++:

Главное меню и пользовательский интерфейс. Управление функциями ассемблера: добавление, удаление и редактирование записей; генерация пароля; шифрование и дешифрование данных.

Логика тайм-аута бездействия.

Используемые прерывания: int 16h, int 21h, int 1Ah.

Ход работы: на рисунке 1 представлено использование различных функций готового программного продукта, которые требуются по условию

Листинг 1 – Исходный код программы

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

#define KEY_SIZE 64

void
reset_clock()
{
    asm{
        xor cx, cx
        xor dx, dx
        mov ah, 01h
        int 1Ah
    }
}

void
one_delay()
{
    asm{
        mov cx, 65000
    }
    delay_loop:
    asm{
        dec cx
        jnz delay_loop
    }
}
```

```
void
make_delay(int value)
{
    for (int i = 0; i < value; ++i)
    {
        one_delay();
    }
}

char
get_symbol()
{
    char
    buffer;
    short
    is_ready = 0;

    for (int i = 0; (i < 750) && (is_ready == 0); ++i)
    {
        make_delay(1);
        asm{
            mov ah, 01h
            int 16h
            jz not_ready
            mov is_ready, 1
        }
        not_ready:
        asm{
            xor ah, ah
        }
    }

    if (is_ready == 1)
    {
        asm{
            xor ah, ah
            int 16h
            mov buffer, al
        }
    }
    else
    {
        printf("\nprogram closed due to inactivity\n");
        exit(0);
    }

    return buffer;
}

void
input_(char *str)
{
    char
    buffer = get_symbol();
    int
    offset = 0;

    while ((int)buffer != 13)
    {
        if (((int)buffer == 8) && (offset > 0))
        {
            str[--offset] = '\0';
            printf("%c", 8);
```

```
            printf(" ");
        }
        else if ((int)buffer == 9)
        {
            for (int i = 0; i < 4; ++i, ++offset)
            {
                str[offset] = ' ';
            }
            str[offset] = '\0';
        }
        else if ((int)buffer != 0)
        {
            str[offset++] = buffer;
            str[offset] = '\0';
        }

        if ((int)buffer != 0)
        {
            printf("%c", buffer);
        }
        buffer = get_symbol();
    }

    printf("\n");
}

char
get_random_byte()
{
    char random_byte = '\0';

    asm{
        xor dx, dx
        xor ah, ah
        int 1Ah
        mov bx, dx

        mov ah, 2Ch
        int 21h
        mov ax, bx
        xor ax, dx

        mov random_byte, al
    }

    return random_byte;
}

char
get_random_symbol(char password_mode)
{
    char
    password_symbols[83] =
    {

'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s'
,'t','u','v','w','x','y','z',

'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S'
,'T','U','V','W','X','Y','Z',
        '0','1','2','3','4','5','6','7','8','9',
        '!','@','#','~','%','^','&','*','(',')','-
','+','=','`','/','[',']', '<','>','?'
    };
```

```
    short
    random_byte = 0;

    asm{
        xor dx, dx
        xor ah, ah
        int 1Ah
        mov bx, dx

        mov ah, 2Ch
        int 21h
        mov ax, bx
        xor ax, dx
        xor ah, al

        mov random_byte, ax
    }

    return password_symbols[abs(random_byte % password_mode)];
}

void
generate_password(short password_size, char password_mode, char
**_password)
{
    char
    password[26];
    char
    random_symbol = '\0';

    asm{
        mov di, 0
    }

    for (int i = 0; i < password_size; ++i)
    {
        random_symbol = get_random_symbol(password_mode);

        asm{
            mov bx, random_symbol
            mov password[di], bl

            inc di
        }

        make_delay(30);
        printf("*");
    }
    printf("\n");

    memcpy(*_password, password, password_size);
}

void
generate_password_case(char **password)
{
    short
    password_size = 8;
    char
    password_mode = 's';
    char
    password_size_str[16];
```

```
    printf("enter size of password (from 1 to 25): ");
    while (1)
    {
        input_(password_size_str);
        password_size = atoi(password_size_str);
        rewind(stdin);

        if (password_size > 0 && password_size < 26)
        {
            break;
        }

        printf("wrong input\n");
    }

    printf("l - lowercase letters\n");
    printf("u - lower and uppercase letters\n");
    printf("n - letters and numbers\n");
    printf("s - letters, numbers and symbols\n");
    while (1)
    {
        password_mode = get_symbol();
        printf("%c", password_mode);
        getchar();

        if (password_mode == 'l' || password_mode == 'L')
        {
            password_mode = (char)26;
            break;
        }
        else if (password_mode == 'u' || password_mode == 'U')
        {
            password_mode = (char)52;
            break;
        }
        else if (password_mode == 'n' || password_mode == 'N')
        {
            password_mode = (char)62;
            break;
        }
        else if (password_mode == 's' || password_mode == 'S')
        {
            password_mode = (char)82;
            break;
        }
        else
        {
            printf("wrong input\n");
        }
    }

    generate_password(password_size, password_mode, password);
}

void
initialize_bytes(char **_bytes, char *_key)
{
    short
    key_size = KEY_SIZE;
    char
    bytes[256];
    char
    key[256];
```

```
    memcpy(key, _key, key_size * sizeof(char));

    for (int i = 0; i < 256; ++i)
    {
        bytes[i] =  i;
    }

    asm{
        xor si, si
        xor di, di
    }
    second_init_1:
    asm{
        mov bl, bytes[si]
        xor bh, bh
        add di, bx
        push si

        cmp si, key_size
        jb second_init_2
    }
    get_module_1:
    asm{
        sub si, key_size
        cmp si, key_size
        jnb get_module_1
    }
    second_init_2:
    asm{
        mov bl, key[si]
        xor bh, bh
        add di, bx
        pop si

        cmp di, 256
        jb swap_elements
    }
    get_module_2:
    asm{
        sub di, 256
        cmp di, 256
        jnb get_module_2
    }
    swap_elements:
    asm{
        mov bh, bytes[si]
        mov bl, bytes[di]
        mov bytes[si], bl
        mov bytes[di], bh

        inc si
        cmp si, 256
        jb second_init_1
    }

    memcpy(*_bytes, bytes, 256 * sizeof(char));
}

char
get_key_byte(char **_bytes, short *_x, short *_y)
{
    char
    key_byte;
    char
```

```
bytes[256];
short
x = *_x;
short
y = *_y;

memcpy(bytes, *_bytes, 256 * sizeof(char));

asm{
    inc x
    cmp x, 256
    jb initialize_y
}
module_of_x:
asm{
    sub x, 256
    cmp x, 256
    jnb module_of_x
}

initialize_y:
asm{
    mov si, x
    mov dl, bytes[si]
    xor dh, dh
    add y, dx

    cmp y, 256
    jb swap_and_return
}
module_of_y:
asm{
    sub y, 256
    cmp y, 256
    jnb module_of_y
}

swap_and_return:
asm{
    mov si, x
    mov di, y

    mov dl, bytes[si]
    mov cl, bytes[di]
    mov bytes[si], cl
    mov bytes[di], dl

    xor dh, dh
    xor ch, ch

    add dx, cx

    cmp dx, 256
    jb evaluate
}
module_of_dx:
asm{
    sub dx, 256
    cmp dx, 256
    jnb module_of_dx
}

evaluate:
asm{
```

```
        mov si, dx
        mov dl, bytes[si]
        mov key_byte, dl
    }

    *_x = x;
    *_y = y;
    memcpy(*_bytes, bytes, 256 * sizeof(char));

    return key_byte;
}

void
bytes_encode(char *data, char **bytes, char **result)
{
    short
    data_size = strlen(data);
    char
    *cipher = (char *) calloc(data_size, sizeof(char));
    char
    key_byte, data_byte, encoded_byte;
    short
    x = 0, y = 0;

    for (short i = 0; i < data_size; ++i)
    {
        key_byte = get_key_byte(bytes, &x, &y);
        data_byte = data[i];

        asm{
            mov si, i
            mov dh, data_byte
            mov dl, key_byte

            xor dh, dl
            mov encoded_byte, dh
        }

        cipher[i] = encoded_byte;
    }

    memcpy(*result, cipher, data_size);

    free(cipher);
}

void
encode(char *data, char *key, char **cipher)
{
    size_t
    data_size = strlen(data);
    char
    *bytes = (char *) calloc(256, sizeof(char));

    initialize_bytes(&bytes, key);

    char
    *_cipher = (char *) calloc(data_size, sizeof(char));

    bytes_encode(data, &bytes, &_cipher);

    memcpy(*cipher, _cipher, data_size);

    free(bytes), free(_cipher);
```

```
}

void
decode(char *cipher, char *key, char **data)
{
    encode(cipher, key, data);
}

void
generate_key(char **key)
{
    printf("generating key");

    for (int i = 0; i < KEY_SIZE; ++i)
    {
        (*key)[i] = get_random_byte();
        make_delay(8);
        if (i % 8 == 0)
        {
            printf(".");
        }
    }

    printf("\n");
}

void
add_key(char *key)
{
    FILE
    *file = fopen("KEYS.txt", "a+b");

    while (file == NULL)
    {
        /*printf("can't open the file");
        return;*/
        file = fopen("KEYS.txt", "a+b");
    }

    fseek(file, 0, SEEK_END);

    fwrite(key, sizeof(char), KEY_SIZE, file);

    fclose(file);
}

void
find_key(char *key, int index)
{
    FILE
    *file = fopen("KEYS.txt", "r+b");

    if (file == NULL)
    {
        printf("can't open the file\n");
        return;
    }

    fseek(file, KEY_SIZE * index, SEEK_SET);

    fread(key, sizeof(char), KEY_SIZE, file);

    fclose(file);
}
```

```
void
remove_key(int index)
{
    FILE
    *reader = fopen("KEYS.txt", "r+b"),
    *writer;
    char
    *keys = (char *) calloc(50 * KEY_SIZE, sizeof(char));
    int
    readed = 0;

    if (reader == NULL)
    {
        printf("can't open the file\n");
        return;
    }

    fseek(reader, 0, SEEK_SET);

    while (feof(reader) == 0)
    {
        if (ftell(reader) == index * KEY_SIZE)
        {
            fseek(reader, KEY_SIZE, SEEK_CUR);
        }
        fread(keys + readed, sizeof(char), KEY_SIZE, reader);
        readed += KEY_SIZE;
    }
    fclose(reader);

    writer = fopen("KEYS.txt", "w+b");

    if (writer == NULL)
    {
        printf("can't open the file\n");
        return;
    }

    fwrite(keys, sizeof(char), readed - KEY_SIZE, writer);

    fclose(writer);
    free(keys);
}

void
encode_data(char **values, int amount)
{
    char
    *key = (char *) calloc(KEY_SIZE, sizeof(char));
    generate_key(&key);
    add_key(key);

    for (int i = 0; i < amount; ++i)
    {
        char
        *buffer = (char *) calloc(KEY_SIZE, sizeof(char));
        encode(values[i], key, &buffer);
        memcpy(values[i], buffer, strlen(buffer));
        free(buffer);
    }

    free(key);
}
```

```
void
encode_data(char **values, int amount, char *key)
{
    for (int i = 0; i < amount; ++i)
    {
        char
        *buffer = (char *) calloc(KEY_SIZE, sizeof(char));
        encode(values[i], key, &buffer);
        memcpy(values[i], buffer, strlen(buffer));
        free(buffer);
    }
}

void
get_values_array(char ***values, char **encoded, int amount)
{
    for (int i = 0; i < amount; ++i)
    {
        sprintf
        (
            (*values)[i],

"[%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%
u,%u,%u,%u,%u,%u,%u,%u]",
            (unsigned char)encoded[i][0],  (unsigned char)encoded[i][1],
            (unsigned char)encoded[i][2],  (unsigned char)encoded[i][3],
            (unsigned char)encoded[i][4],  (unsigned char)encoded[i][5],
            (unsigned char)encoded[i][6],  (unsigned char)encoded[i][7],
            (unsigned char)encoded[i][8],  (unsigned char)encoded[i][9],
            (unsigned char)encoded[i][10], (unsigned char)encoded[i][11],
            (unsigned char)encoded[i][12], (unsigned char)encoded[i][13],
            (unsigned char)encoded[i][14], (unsigned char)encoded[i][15],
            (unsigned char)encoded[i][16], (unsigned char)encoded[i][17],
            (unsigned char)encoded[i][18], (unsigned char)encoded[i][19],
            (unsigned char)encoded[i][20], (unsigned char)encoded[i][21],
            (unsigned char)encoded[i][22], (unsigned char)encoded[i][23],
            (unsigned char)encoded[i][24], (unsigned char)encoded[i][25],
            (unsigned char)encoded[i][26], (unsigned char)encoded[i][27],
            (unsigned char)encoded[i][28], (unsigned char)encoded[i][29],
            (unsigned char)encoded[i][30], (unsigned char)encoded[i][31]
        );
    }
}

void
serialize(char **serialized, char **titles, char **encoded_values, char
*name, int amount)
{
    int
    size = 0;

    char
    **values = (char **) calloc(amount, sizeof(char *));
    for (int i = 0; i < amount; ++i)
    {
        values[i] = (char *) calloc(128, sizeof(char));
    }

    get_values_array(&values, encoded_values, amount);

    sprintf(serialized[size++], "%s:\n", name);
    sprintf(serialized[size++], "{\n");
    for (int j = 0; j < amount; ++j)
```

```
    {
        sprintf(serialized[size++], "    %s : %s;\n", titles[j],
values[j]);
    }
    sprintf(serialized[size++], "}\n");

    for (int k = 0; k < amount; ++k)
    {
        free(values[k]);
    }
    free(values);
}

void
save_data(char **serialized, int amount, char *filename)
{
    FILE
    *file = fopen(filename, "a+");

    if (file == NULL)
    {
        printf("can't open the file\n");
        return;
    }

    for (int i = 0; i < amount; ++i)
    {
        fputs(serialized[i], file);
    }

    fclose(file);
}

void
add_other_info(char **password)
{
    char
    **titles = (char **) calloc(10, sizeof(char *)),
    **information = (char **) calloc(10, sizeof(char *));

    for (int i = 1; i < 10; ++i)
    {
        titles[i] = (char *) calloc(32, sizeof(char));
        information[i] = (char *) calloc(32, sizeof(char));
    }

    char
    *_password = "password",
    *_login = "login",
    *name = (char *) calloc(32, sizeof(char)),
    input[16];

    titles[0] = _password;
    information[0] = *password;

    int
    amount = 1;

    printf("enter name of your password: ");
    rewind(stdin);
    input_(name);

    printf("lgn - save password with login, wlg - without\n");
```

```
while (1)
{
    rewind(stdin);
    input_(input);

    if (strcmp(input, "lgn") == 0)
    {
        printf("enter login: ");
        rewind(stdin);
        input_(information[amount]);

        titles[amount++] = _login;
        break;
    }
    else if (strcmp(input, "wlg") == 0)
    {
        break;
    }
    else
    {
        printf("wrong input");
    }
}

while (1)
{
    printf("add - add other fields, skip - stop adding\n");
    rewind(stdin);
    input_(input);

    if (strcmp(input, "add") == 0)
    {
        printf("enter title: ");
        rewind(stdin);
        input_(titles[amount]);

        printf("enter info: ");
        rewind(stdin);
        input_(information[amount++]);
    }
    else if (strcmp(input, "skip") == 0)
    {
        break;
    }
    else
    {
        printf("wrong input\n");
    }

    if (amount > 9)
    {
        printf("there are 10 fields, so adding stopped\n");
    }
}

char
**serialized = (char **) calloc(amount + 5, sizeof(char *));
for (int j = 0; j < amount + 5; ++j)
{
    serialized[j] = (char *) calloc(256, sizeof(char));
}

encode_data(information, amount);
```

```
    serialize(serialized, titles, information, name, amount);

    save_data(serialized, amount + 3, "USER_DAT.txt");
}

void
add_data_menu()
{
    char
    *password = (char *) calloc(32, sizeof(char));

    printf("gen - generate password, slf - enter your password\n");

    while (1)
    {
        char
        input[16];
        rewind(stdin);
        input_(input);

        if (strcmp(input, "gen") == 0)
        {
            generate_password_case(&password);

            break;
        }
        else if (strcmp(input, "slf") == 0)
        {
            printf("enter your password: ");
            rewind(stdin);
            input_(password);

            break;
        }
        else
        {
            printf("wrong input\n");
        }
    }

    add_other_info(&password);

    free(password);
}

long
find_position(char *name, char *filename, int *index)
{
    FILE
    *file = fopen(filename, "r+");
    long
    position = 0;

    if (file == NULL)
    {
        printf("can't open the file\n");
        return -1;
    }
    char
    buffer[256];

    while (feof(file) == 0)
    {
        position = ftell(file);
```

```
        fgets(buffer, 256, file);

        if (memcmp(buffer, name, strlen(name)) == 0)
        {
            fclose(file);
            return position;
        }

        if (buffer[0] != ' ' && buffer[0] != '{' && buffer[0] != '}')
        {
            (*index)++;
        }
    }

    fclose(file);
    return -1;
}

void
deserialize(char *line, char *title, char *value)
{
    int
    *encoded = (int *) calloc(32, sizeof(int));

    sscanf
    (
        line,
        "    %s :
[%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u
,%u,%u,%u,%u,%u,%u,%u];\n",
        title,
        &encoded[0],  &encoded[1],
        &encoded[2],  &encoded[3],
        &encoded[4],  &encoded[5],
        &encoded[6],  &encoded[7],
        &encoded[8],  &encoded[9],
        &encoded[10], &encoded[11],
        &encoded[12], &encoded[13],
        &encoded[14], &encoded[15],
        &encoded[16], &encoded[17],
        &encoded[18], &encoded[19],
        &encoded[20], &encoded[21],
        &encoded[22], &encoded[23],
        &encoded[24], &encoded[25],
        &encoded[26], &encoded[27],
        &encoded[28], &encoded[29],
        &encoded[30], &encoded[31]
    );

    for (int i = 0; i < 32; ++i)
    {
        value[i] = (char)encoded[i];
    }

    free(encoded);
}

void
decode_data(char **encoded_values, int index, int amount)
{
    char
    *key = (char *) calloc(KEY_SIZE, sizeof(char));
    find_key(key, index);
```

```
    for (int i = 0; i < amount; ++i)
    {
        char
        *buffer = (char *) calloc(64, sizeof(char));
        decode(encoded_values[i], key,  &buffer);
        memcpy(encoded_values[i], buffer, strlen(buffer));
    }

    free(key);
}

void
write_output(char *name, char **titles, char **values, int amount)
{
    FILE
    *file = fopen("OUTPUT.txt", "w+");

    if (file == NULL)
    {
        printf("can't write output to file\n");
        return;
    }

    fprintf(file, "%s :\n", name);
    fprintf(file, "{\n");
    for (int i = 0; i < amount; ++i)
    {
        fprintf(file, "    %s : %s\n", titles[i], values[i]);
    }
    fprintf(file, "}\n");

    fclose(file);
}

void
get_data(char *name, char **titles, char **values, int *amount, char
*filename)
{
    FILE
    *file = fopen(filename, "r+");

    if (file == NULL)
    {
        printf("can't open the file\n");
        return;
    }

    int
    index = 0;
    long
    position = find_position(name, filename, &index);

    if (position == -1)
    {
        printf("this data is not found\n");
        fclose(file);
        return;
    }

    fseek(file, position, SEEK_CUR);

    int
    size = 0;
    char
```

```
    *buffer = (char *) calloc(256, sizeof(char));
    fgets(buffer, 256, file);
    fgets(buffer, 256, file);

    while (feof(file) == 0)
    {
        fgets(buffer, 256, file);

        if (strcmp(buffer, "}\n") == 0)
        {
            break;
        }

        deserialize(buffer, titles[size], values[size]);
        ++size;
    }

    fclose(file);

    decode_data(values, index, size);

    *amount = size;

    free(buffer);
}

void
get_data_menu()
{
    char
    *name = (char *) calloc(32, sizeof(char)),
    **titles = (char **) calloc(10, sizeof(char *)),
    **values = (char **) calloc(10,  sizeof(char *));

    int
    amount = 0;

    for (int i = 0; i < 10; ++i)
    {
        titles[i] = (char *) calloc(64, sizeof(char)),
        values[i] = (char *) calloc(64, sizeof(char));
    }

    printf("name of password: ");
    rewind(stdin);
    input_(name);

    get_data(name, titles, values, &amount, "USER_DAT.txt");

    for (int j = 0; j < amount; ++j)
    {
        printf("name of title: %s, value: %s\n", titles[j], values[j]);
    }
    write_output(name, titles, values, amount);

    printf("this data was also written into file [%s], but after another
output this data will be overwritten.\n", "OUTPUT.txt");

    for (int k = 0; k < 10; ++k)
    {
        free(titles[k]), free(values[k]);
    }
    free(titles), free(values), free(name);
}
```

```
void
rewrite_data(char **serialized, int amount, char *name, char *filename)
{
    FILE
    *reader = fopen(filename, "r+"),
    *writer;

    if (reader == NULL)
    {
        printf("can't open the file\n");
        return;
    }

    int
    index = -1,
    size = 0;
    long
    position = find_position(name, filename, &index);

    char
    **lines = (char **) calloc(500, sizeof(char *)),
    *buffer = (char *) calloc(256, sizeof(char));
    for (int i = 0; i < 500; ++i)
    {
        lines[i] = (char *) calloc(256, sizeof(char));
    }

    while (feof(reader) == 0)
    {
        if (ftell(reader) != position)
        {
            fgets(lines[size++], 256, reader);
        }
        else
        {
            fgets(buffer, 256, reader);
            fgets(buffer, 256, reader);

            while (buffer[0] == '{' || buffer[0] == '}' || buffer[0] == '
')
            {
                buffer = (char *) calloc(256, sizeof(char));
                fgets(buffer, 256, reader);
                if (feof(reader) != 0)
                {
                    break;
                }
            }

            for (int j = 0; j < amount; ++j, ++size)
            {
                memcpy(lines[size], serialized[j], strlen(serialized[j]));
            }

            if (feof(reader) == 0)
            {
                memcpy(lines[size++], buffer, strlen(buffer));
            }
        }
    }

    fclose(reader);
    writer = fopen(filename, "w+");
```

```
    if (writer == NULL)
    {
        printf("can't open the file (rewrite_data, 2)\n");
        return;
    }

    for (int k = 0; k < size; ++k)
    {
        fputs(lines[k], writer);
    }

    fclose(writer);

    for (int l = 0; l < 500; ++l)
    {
        free(lines[l]);
    }
    free(lines), free(buffer);
}

void
edit_data_menu()
{
    char
    *name = (char *) calloc(32, sizeof(char)),
    **titles = (char **) calloc(10, sizeof(char *)),
    **values = (char **) calloc(10,  sizeof(char *)),
    **serialized = (char **) calloc(20, sizeof(char *)),
    *key = (char *) calloc(KEY_SIZE, sizeof(char)),
    input[16];

    int
    amount = 0,
    index = 0;

    for (int i = 0; i < 10; ++i)
    {
        titles[i] = (char *) calloc(64, sizeof(char)),
        values[i] = (char *) calloc(64, sizeof(char)),
        serialized[i] = (char *) calloc(256, sizeof(char)),
        serialized[i + 10] = (char *) calloc(256, sizeof(char));
    }

    printf("name of password to edit: ");
    rewind(stdin);
    input_(name);

    get_data(name, titles, values, &amount, "USER_DAT.txt");

    for (int j = 0; j < amount; ++j)
    {
        printf("name of title: %s, value: %s\n", titles[j], values[j]);

        while (1)
        {
            printf("edt - edit title, edv - edit value, skip - don't
edit\n");
            rewind(stdin);
            input_(input);

            if (strcmp(input, "edt") == 0)
            {
                printf("enter new title: ");
```

```
                rewind(stdin);
                input_(titles[j]);
            }
            else if (strcmp(input, "edv") == 0)
            {
                printf("enter new value: ");
                rewind(stdin);
                input_(values[j]);
            }
            else if (strcmp(input, "skip") == 0)
            {
                break;
            }
            else
            {
                printf("wrong input\n");
            }
        }
    }

    (void) find_position(name, "USER_DAT.txt", &index);
    find_key(key, index);
    encode_data(values, amount, key);
    serialize(serialized, titles, values, name, amount);
    amount += 3;
    rewrite_data(serialized, amount, name, "USER_DAT.txt");

    for (int k = 0; k < 10; ++k)
    {
        free(titles[k]), free(values[k]),
        free(serialized[k]), free(serialized[k + 10]);
    }
    free(titles), free(values), free(serialized),
    free(key), free(name);
}

void
remove_data_menu()
{
    char
    *name = (char *) calloc(32, sizeof(char));
    int
    index = 0;

    printf("enter name of password to remove: ");
    rewind(stdin);
    input_(name);

    if (find_position(name, "USER_DAT.txt", &index) == -1)
    {
        printf("this name is not found\n");
        return;
    }

    remove_key(index);
    rewrite_data(NULL, 0,  name, "USER_DAT.txt");

    free(name);
}

int
main()
{
    char
```

```
    input[16];
    while (1)
    {
        printf("add - add password,\nedit - edit password,\nrmv - remove
password,\nget - print,\nexit - exit\n");
        rewind(stdin);
        input_(input);
        if (strcmp(input, "add") == 0)
        {
            add_data_menu();
        }
        else if (strcmp(input, "edit") == 0)
        {
            edit_data_menu();
        }
        else if (strcmp(input, "rmv") == 0)
        {
            remove_data_menu();
        }
        else if (strcmp(input, "get") == 0)
        {
            get_data_menu();
        }
        else if (strcmp(input, "exit") == 0)
        {
            break;
        }
        else
        {
            printf("wrong input\n");
        }
    }
      return 0;
}
```

```
exit - exit
get
name of password: VK
name of title: password, value: VladStepanoff2021
name of title: login, value: Vladichka
name of title: AccauntCreatingDate, value: 20.12.2023
this data was also written into file [OUTPUT.txt], but after another output this
 data will be overwritten.
add - add password,
edit - edit password,
rmv - remove password,
get - print,
exit - exit
EXIT
wrong input
add - add password,
edit - edit password,
rmv - remove password,
get - print,
exit - exit
exit
```


Рисунок 1 – Использование различных функций готового программного продукта

Выводы: в результате лабораторной работы была выполнена поставленная задача с использованием прерываний 21h, 16h, 1Ah