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,%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,%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;

}

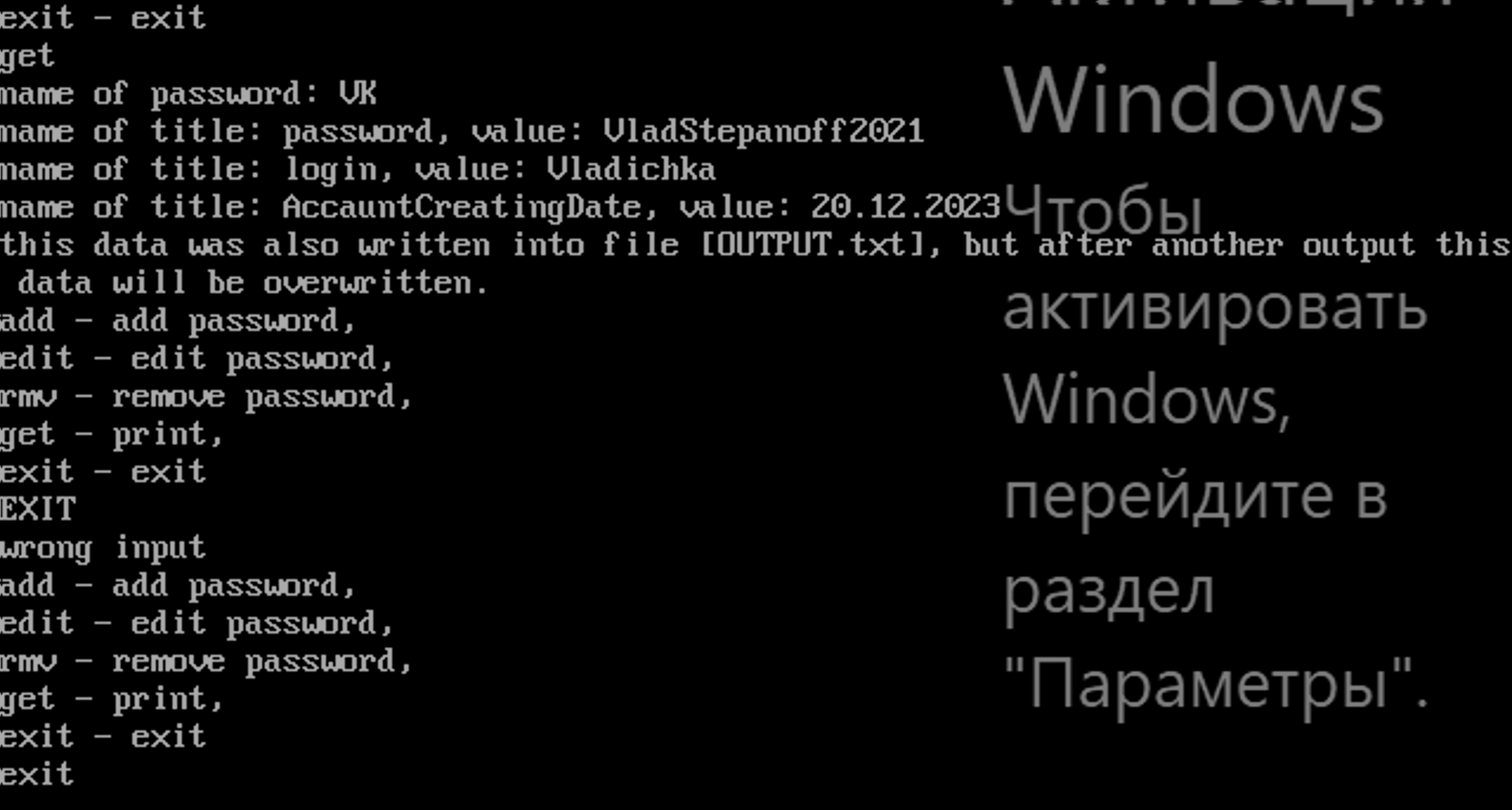


Рисунок 1 – Использование различных функций готового программного продукта

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