Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы алгоритмизации и

программирования

УЧЕБНАЯ ПРАКТИКА

Студент Е.Е. Алборов

Руководитель П. А. Дулько

МИНСК 2022

**СОДЕРЖАНИЕ**

1. Введение ……………………………………………………………………....3

2. Общие сведения о хэш-функции…………………………………………….4

2.1 Описание алгоритма реализации   
хэш-функции………………………………………………………………….…5

3 Блок-схема хэш-функции..…………………………………………………....7

3.1 Основная функция (main)…………………………………………………..9

4. Заключение…………………………………………………………………..15

5. Листинг кода………………………………………………………………....16

ВВЕДЕНИЕ

Данная работа заключается в вычислении хэш-суммы логина и пароля и проверки авторизации. В данной работе будет показан алгоритм реализации хэш-функции. Используемая структура данных- сбалансированное бинарное дерево поиска .Также тут будут приведены блок-схемы для функций и листинг кода на языке Си.

**Общие сведения о хэш-функции**

**Криптографическая хеш-функция** - это математический алгоритм, который отображает данные произвольного размера(например логин и пароль) в битовый массив фиксированного размера(конечный результат).

Результат, производимый хеш-функцией, называется **«хэш-суммой»** или же просто **«хэшем»,** а входные данные часто называют **«сообщением».  
  
Соль-** Используется для усложнения определения прообраза хэш-функции. Различают статическую соль (одна и та же для всех входных значений) и динамическую (генерируется для каждого входного значения персонально).

Свойства хэш-функции:

a) хэш-функция является детерминированной, то есть одно и то же сообщение приводит к одному и тому же хэш-значению.

b) невозможно найти сообщение, которое дает заданное хэш-значение.

c)Практически невозможно найти два разных сообщения с одинаковым  
 хэш-значением(если же hash(m\_1)=hash(m\_2) , то такая пара сообщений m\_1и m\_2называется **коллизией хеш-функции**)

d) небольшое изменение в сообщении изменяет хэш очень сильно.

**Описание алгоритма реализации хэш-функции**

Входными параметрами хэш-функции будут:  
1) логин пользователя  
2) пароль пользователя  
3)длина требуемой хэш-суммы.

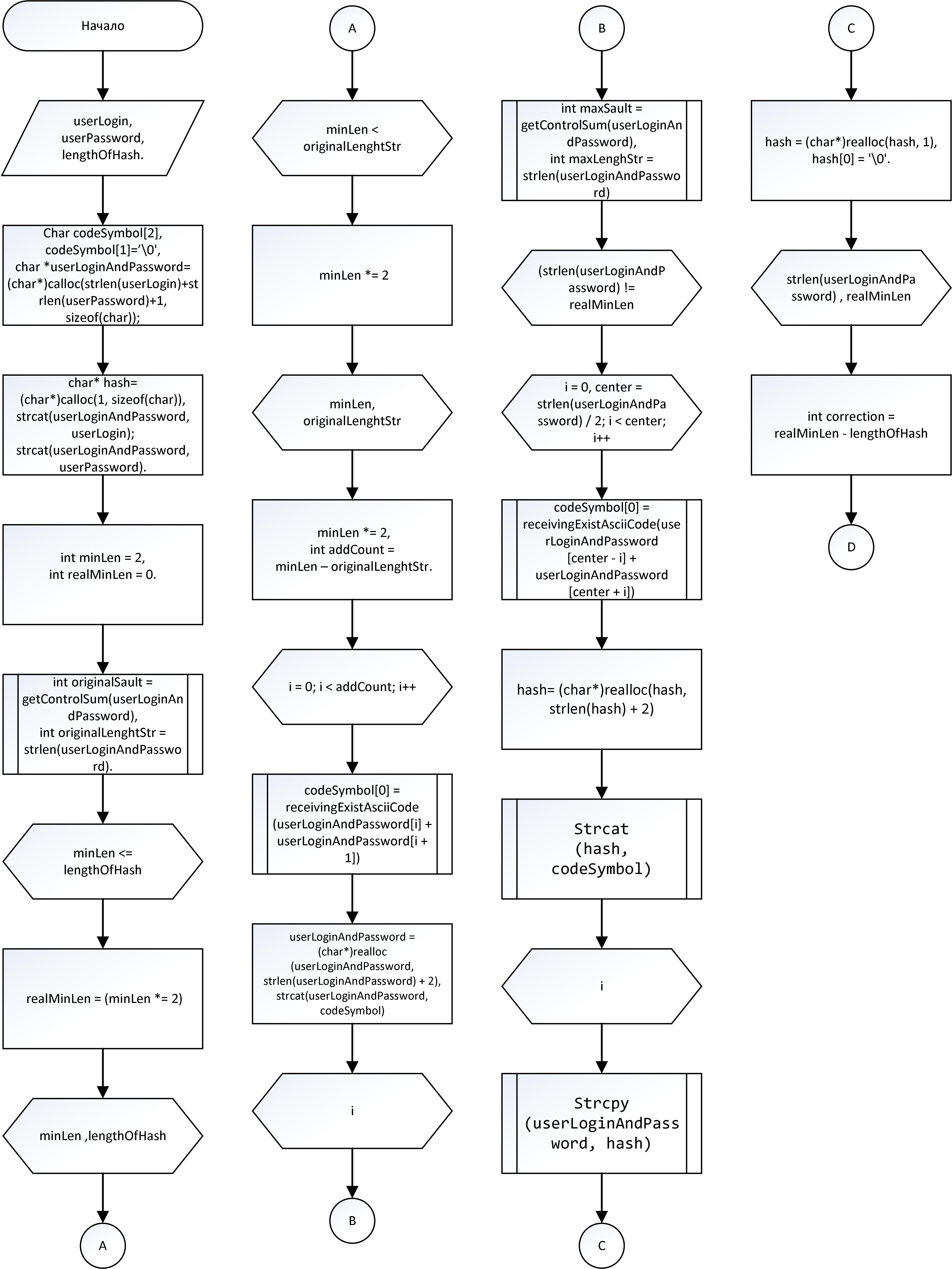
A)  
В созданном массиве символов объединим логин и пароль пользователя, которые являются тоже массивами символов (то есть строками).  
Получаем переменную: char\*userLoginAndPassword;  
А также создадим две переменные:  
int minLen=2;  
int originalLenghtStr=0;  
  
Эти две переменные нужны, так как данная хэш-функция вычисляет  
хэш-сумму произвольной длины, поэтому эти переменные нужны для дальнейших преобразований char\*userLoginAndPassword;  
  
Создадим char\* hash, где будет конечная хэш-сумма.  
  
B)  
Получаем соль оригинальной строки originalSault с помощью функции: getControlSum;  
И длину строки userLoginAndPassword(тоже соль);

C)  
Получаем длину строки, кратной степени двух, ближайшей к заданной длине хэша. То есть while (minLen <= lengthOfHash) realMinLen = (minLen \*= 2);  
(realMinLen пригодится при “сжатии” и приведении к нужной длине)  
  
Далее получим ближайшую к длине исходной строки число такого типа 2^n   
while (minLen < originalLenghtStr) minLen \*= 2;  
(это на случай, если длина логина и пароля будут больше длины хэша)  
  
Делаем длину строки хэша, как минимум, в 2 раза длинней оригинальной строки. minLen \*= 2;  
  
  
  
  
  
  
D)  
Получаем кол-во символов, которое необходимо добавить к строке.  
int addCount = minLen - originalLenghtStr;  
  
Теперь с помощью цикла for (int i = 0; i < addCount; i++) , используя функцию **receivingExistAsciiCode (userLoginAndPassword[i] + userLoginAndPassword[i + 1]);** дополним строку userLoginAndPassword;  
  
E)  
Получаем “максимальную соль”  
int maxSault = getControlSum(userLoginAndPassword);  
int maxLenghStr = strlen(userLoginAndPassword);  
  
F)  
Определение степени сжатия при помощи циклов   
while (strlen(userLoginAndPassword) != realMinLen);  
for (int i = 0, center = strlen(userLoginAndPassword) / 2; i < center; i++);  
  
И функции   
receivingExistAsciiCode(userLoginAndPassword[center - i] + userLoginAndPassword[center + i]);  
Значения которой будем заносить в hash.  
  
Потом перезапишем hash в userLoginAndPassword(для цикла)

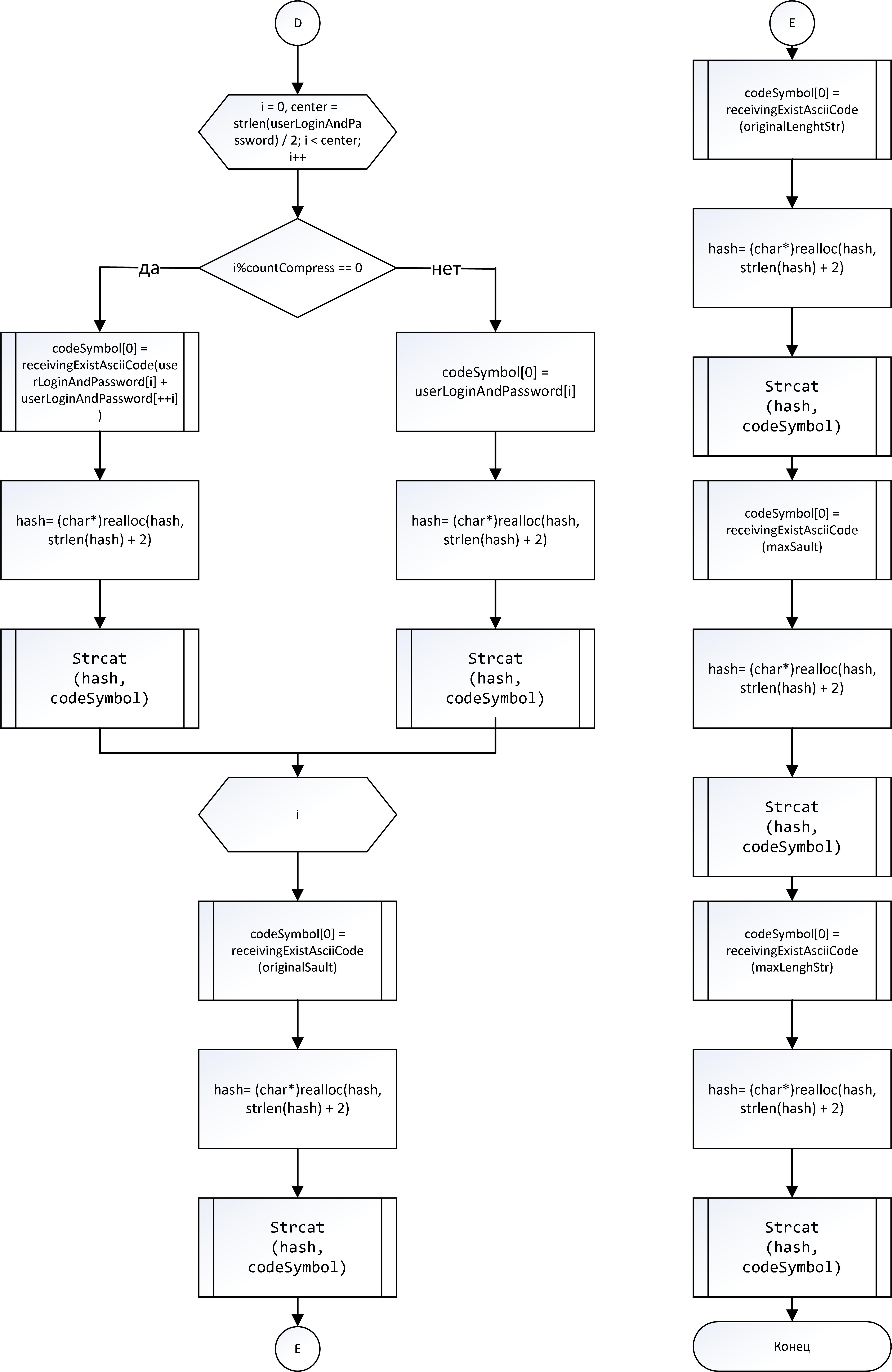
G)  
Здесь будет происходить приведение хэша к нужной длине при помощи цикла for (int i = 0, countCompress = realMinLen / correction; strlen(hash) < (lengthOfHash - 4); i++)

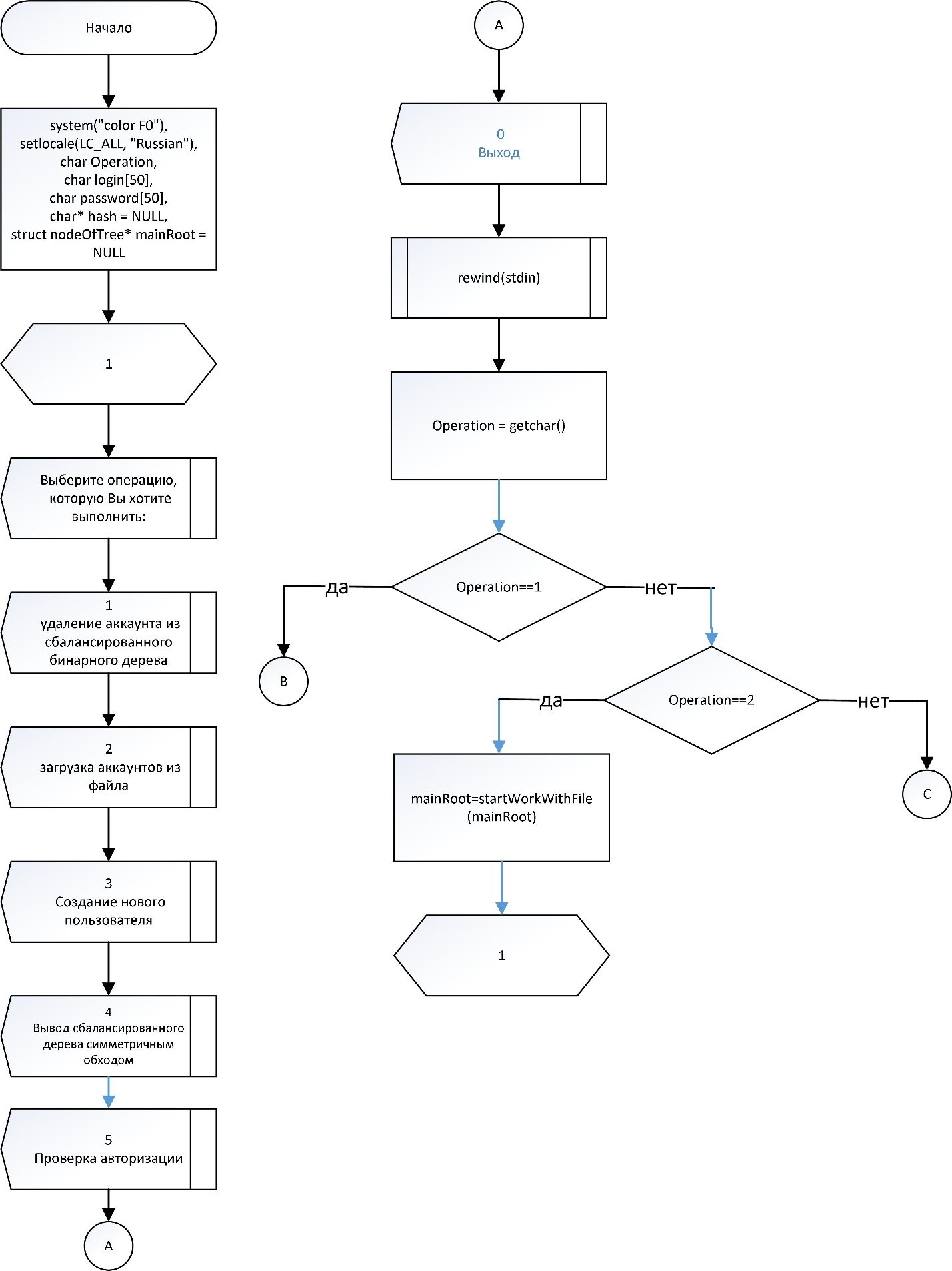
И затем добавление солей:  
receivingExistAsciiCode(originalSault);   
receivingExistAsciiCode(originalLenghtStr);   
receivingExistAsciiCode(maxSault);   
receivingExistAsciiCode(maxLenghStr);   
  
мы получили хэш-сумму hash;

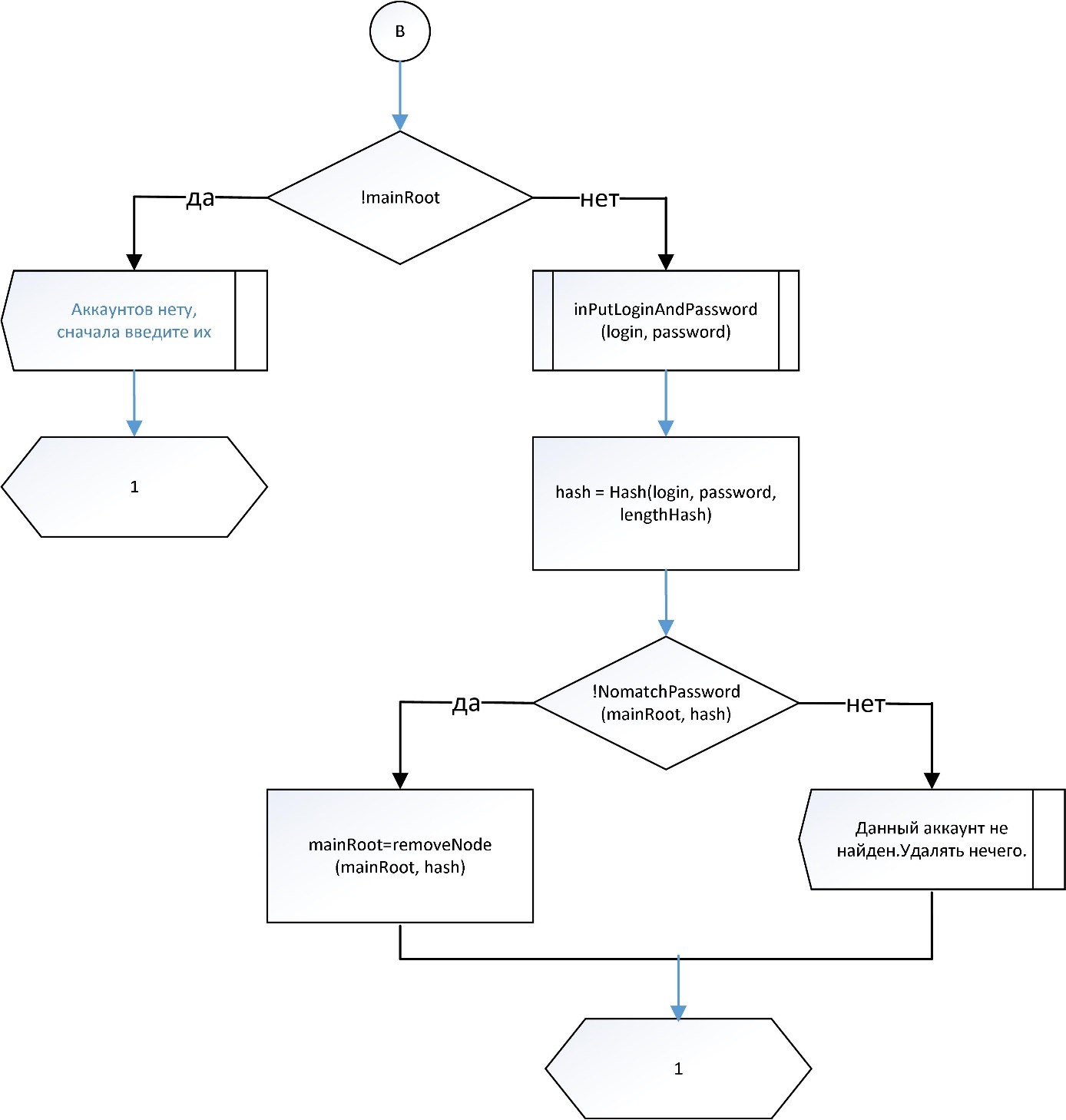
**Блок-схема хэш-функции**

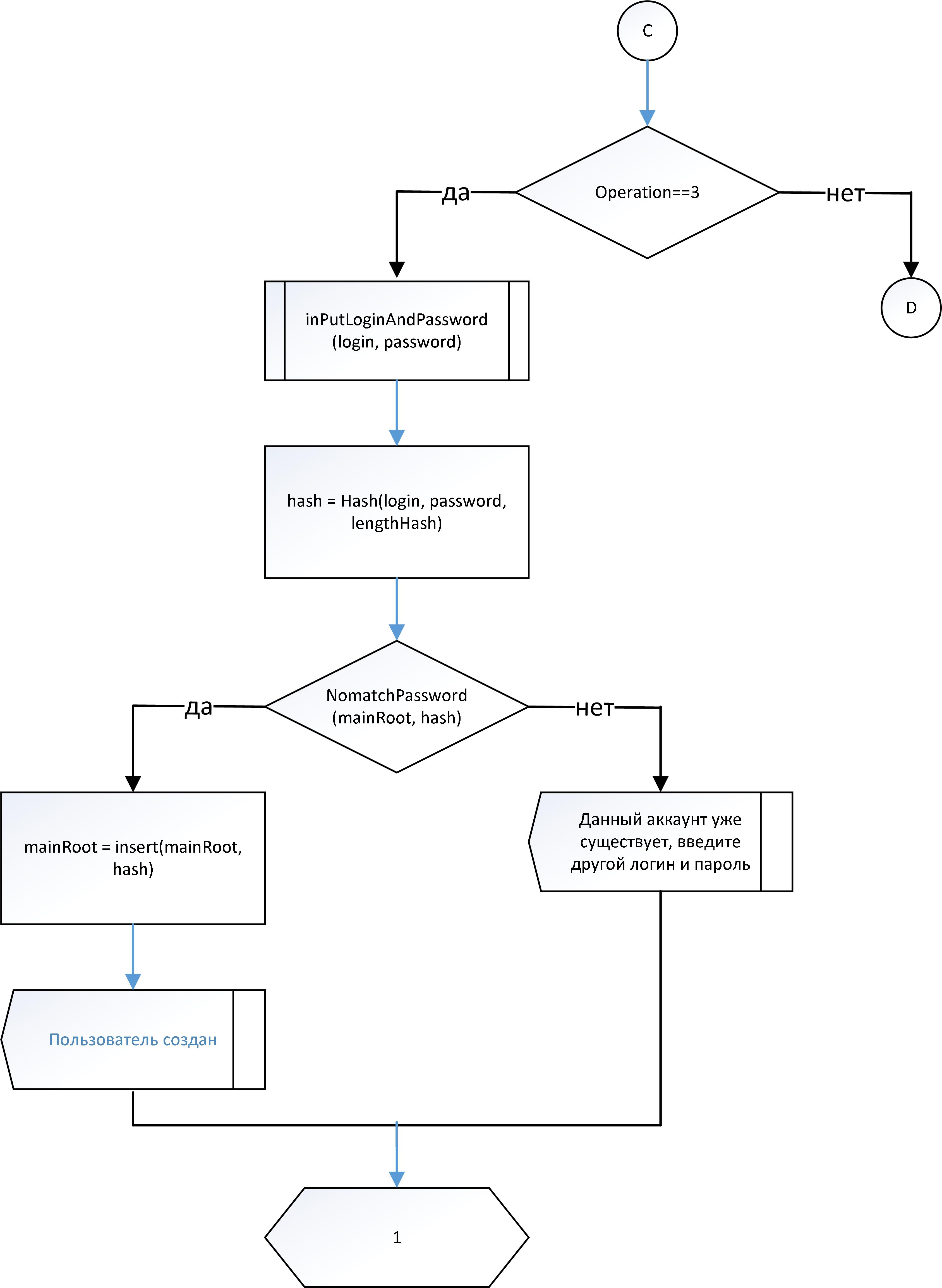


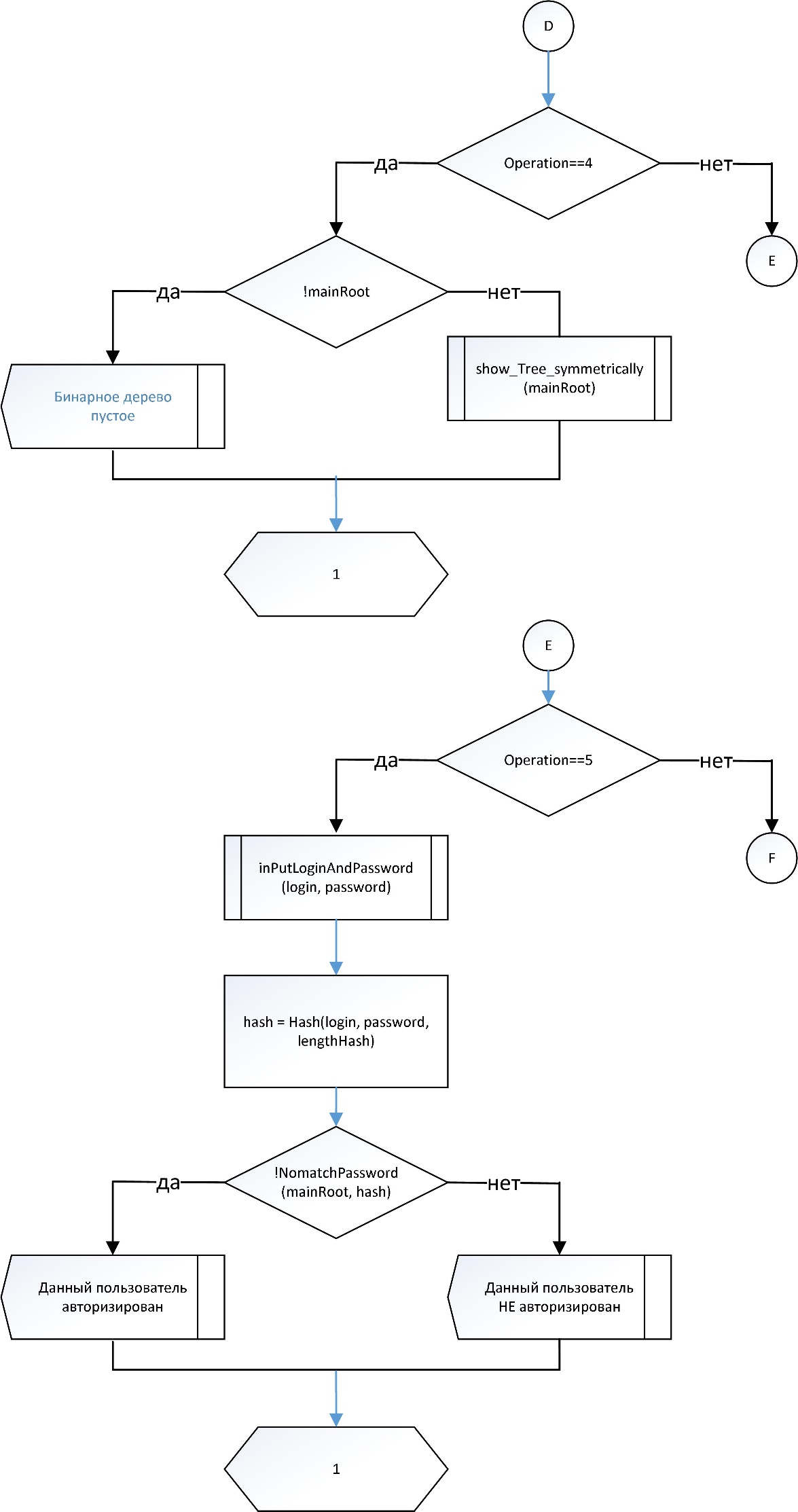
**Основная функция (main)**

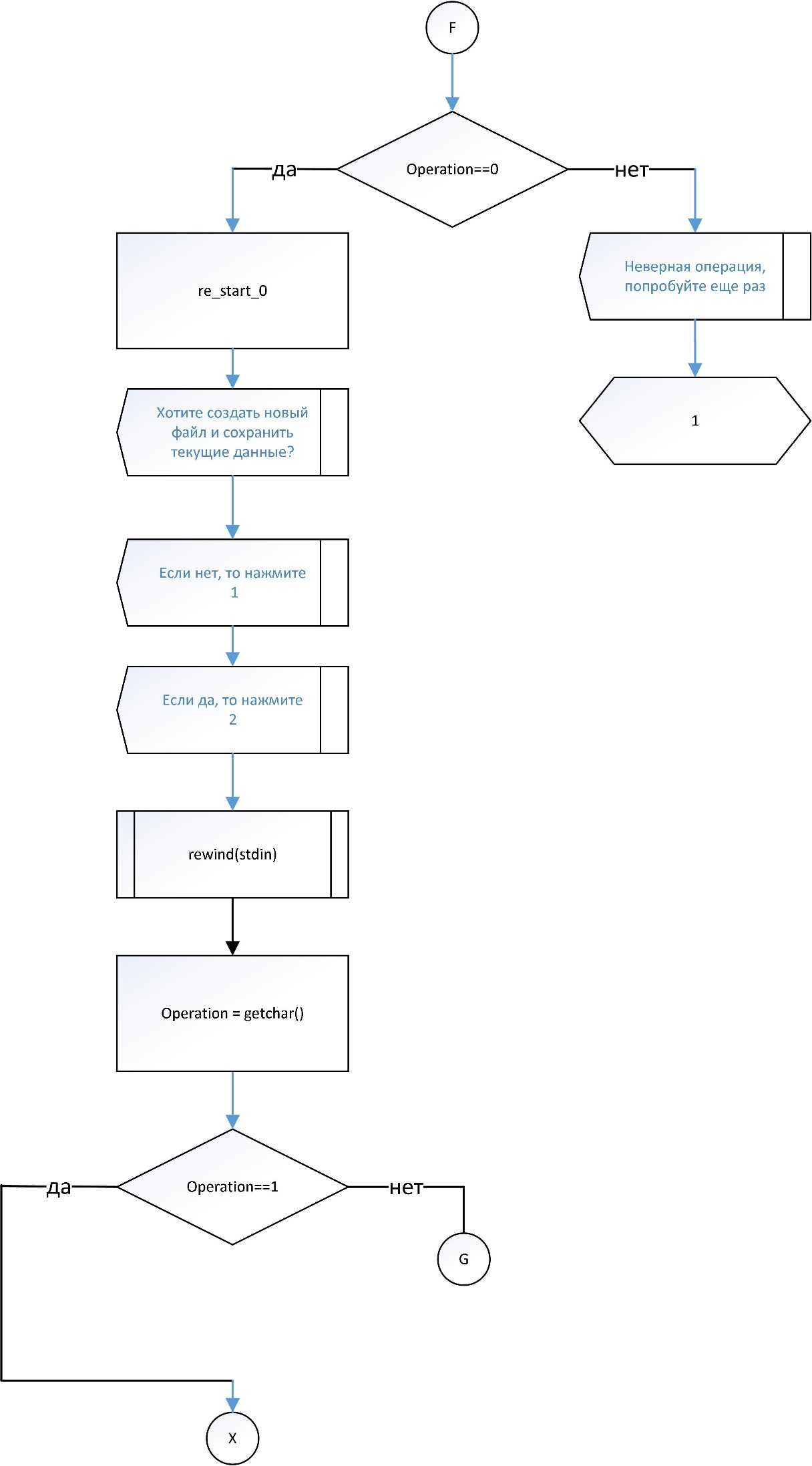


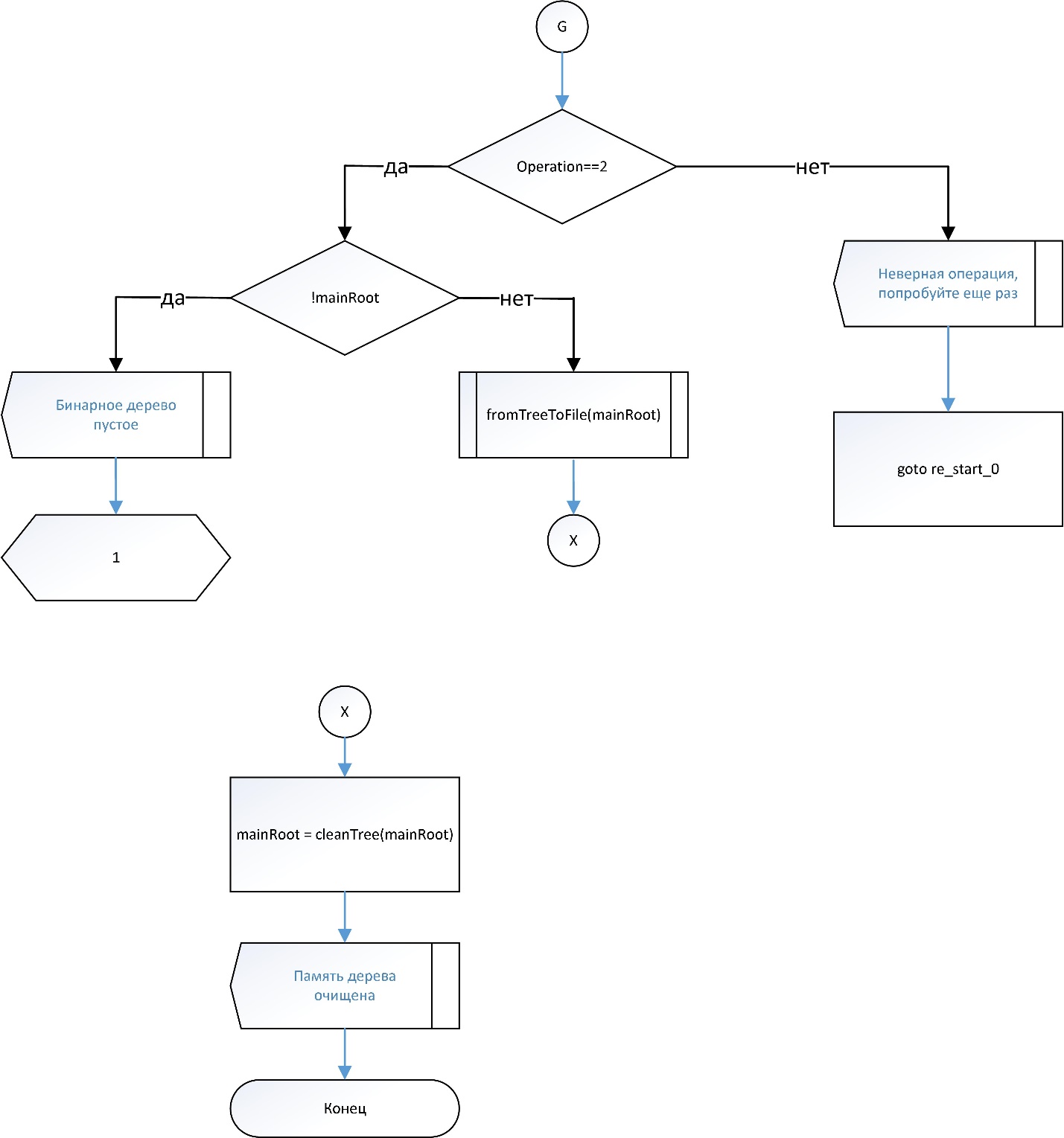
****

****



****

****

****

ЗАКЛЮЧЕНИЕ

**В ходе учебной практики было выяснено, как работать с балансированным бинарным деревом поиска, был рассмотрен алгоритм хэш-функции, а также была проделана работа с файлами.**

ЛИСТИНГ КОДА

**//17.Программа для вычисления хэш-суммы логина и пароля. И проверка авторизации**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#define lengthHash 12**

**#define maxLengthFile 30**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <conio.h>**

**#include <locale.h>**

**#include <Windows.h>**

**struct nodeOfTree**

**{**

**char hash[lengthHash+1];**

**int height;**

**int repeats;**

**struct nodeOfTree\* left;**

**struct nodeOfTree\* right;**

**};**

**char\* Hash(char\* userLogin, char\* userPassword, int lengthOfHash);**

**char receivingExistAsciiCode(int x);**

**int getControlSum(char\*string);**

**int height(struct nodeOfTree\* node);**

**int bFactor(struct nodeOfTree\* node);**

**void fixHeight(struct nodeOfTree\* node);**

**struct nodeOfTree\* rotateright(struct nodeOfTree\* node);**

**struct nodeOfTree\* rotateleft(struct nodeOfTree\* node);**

**struct nodeOfTree\* balance(struct nodeOfTree\* node);**

**struct nodeOfTree\* insert(struct nodeOfTree\* root, char\* keySecondName);**

**struct nodeOfTree\* findmin(struct nodeOfTree\* mainRoot);**

**struct nodeOfTree\* removemin(struct nodeOfTree\* node);**

**struct nodeOfTree\* removeNode(struct nodeOfTree\* mainRoot, char\* removableHash);**

**void show\_Tree\_symmetrically(struct nodeOfTree\* root);**

**struct nodeOfTree\* cleanTree(struct nodeOfTree\* root);**

**void CorrectInput(int\* Information);**

**void inPutLoginAndPassword(char\*login, char\* password);**

**struct nodeOfTree\* startWorkWithFile(struct nodeOfTree\* mainRoot);**

**char\*\* readFromFile(char\*\* accaunts, int\*count);**

**char\*\* fromTreeToFile(struct nodeOfTree\* root);**

**void enumeration(struct nodeOfTree\* root, FILE\*file);**

**void main()**

**{**

**system("color F0");**

**setlocale(LC\_ALL, "Russian");**

**SetConsoleCP(1251);// установка кодовой страницы win-cp 1251 в поток ввода**

**SetConsoleOutputCP(1251); // установка кодовой страницы win-cp 1251 в поток вывода**

**char Operation;**

**char login[50];**

**char password[50];**

**char\*hash;**

**hash = NULL;**

**struct nodeOfTree\* mainRoot = NULL;**

**while (1)**

**{**

**start:**

**printf("\nВыберите операцию, которую Вы хотите выполнить:\n");**

**puts(" 1 удаление аккаунта из сбалансированного бинарного дерева");**

**puts(" 2 загрузка аккаунтов из файла");**

**puts(" 3 Создание нового пользователя");**

**puts(" 4 Вывод сбалансированного дерева симметричным обходом");**

**puts(" 5 Проверка авторизации");**

**puts(" 0 Выход");**

**rewind(stdin);**

**Operation = getchar();**

**switch (Operation)**

**{**

**case'1':**

**if (!mainRoot)**

**{**

**printf("\nАккаунтов нету, сначала введите их");**

**break;**

**}**

**inPutLoginAndPassword(login, password);**

**hash = Hash(login, password, lengthHash);**

**if (!NomatchPassword(mainRoot, hash))**

**mainRoot=removeNode(mainRoot, hash);**

**else**

**printf("\nДанный аккаунт не найден.Удалять нечего.");**

**break;**

**case'2': mainRoot=startWorkWithFile(mainRoot); break;**

**case'3':**

**re\_creation:**

**inPutLoginAndPassword(login, password);**

**hash = Hash(login, password, lengthHash);**

**if (NomatchPassword(mainRoot, hash))**

**{**

**mainRoot = insert(mainRoot, hash);**

**printf("\nПользователь создан");**

**}**

**else**

**{**

**printf("\nДанный аккаунт уже существует, введите другой логин и пароль");**

**goto re\_creation;**

**}**

**break;**

**case'4':**

**if (!mainRoot)**

**{**

**printf("\nБинарное дерево пустое\n");**

**goto start;**

**}**

**show\_Tree\_symmetrically(mainRoot);**

**break;**

**case'5':**

**if (!mainRoot)**

**{**

**printf("\nАккаунтов нету, сначала введите их");**

**break;**

**}**

**inPutLoginAndPassword(login, password);**

**hash = Hash(login, password, lengthHash);**

**if (!NomatchPassword(mainRoot, hash))**

**printf("\nДанный пользователь авторизирован");**

**else**

**printf("\nДанный пользователь НЕ авторизирован");**

**break;**

**case '0':**

**re\_start\_0:**

**printf("\nХотите создать новый файл и сохранить текущие данные?");**

**printf("\nЕсли нет, то нажмите 1");**

**printf("\nЕсли да, то нажмите 2\n");**

**rewind(stdin);**

**Operation = getchar();**

**switch (Operation)**

**{**

**case'1': break;**

**case'2':**

**if (!mainRoot)**

**{**

**printf("\nБинарное дерево пустое\n");**

**goto start;**

**}**

**fromTreeToFile(mainRoot);**

**break;**

**default:printf("\nНеверная операция, попробуйте еще раз"); goto re\_start\_0;**

**}**

**mainRoot = cleanTree(mainRoot);**

**printf("\nПамять дерева очищена");**

**return;**

**default:printf("\nНеверная операция, попробуйте еще раз"); goto start;**

**}**

**}**

**}**

**int height(struct nodeOfTree\* node)**

**{**

**return node ? node->height : 0; //тернарный оператор, если node!=NULL,то глубина, иначе 0**

**}**

**int bFactor(struct nodeOfTree\* node)**

**{**

**if (node->right == NULL && node->left != NULL)**

**{**

**return (height(node->right) - height(node->left)) - 1;**

**}**

**if (node->right != NULL && node->left == NULL)**

**{**

**return (height(node->right) - height(node->left)) + 1;**

**}**

**if (node->right != NULL && node->left != NULL)**

**{**

**return (height(node->right) - height(node->left));**

**}**

**}**

**void fixHeight(struct nodeOfTree\* node)**

**{**

**int leftHeightOfNode = height(node->left);**

**int rightHeightOfNode = height(node->right);**

**node->height = (leftHeightOfNode > rightHeightOfNode ? leftHeightOfNode : rightHeightOfNode) + 1;**

**}**

**struct nodeOfTree\* rotateright(struct nodeOfTree\* node)**

**{**

**struct nodeOfTree\* leftNode = node->left;**

**node->left = leftNode->right;**

**leftNode->right = node;**

**fixHeight(node);**

**fixHeight(leftNode);**

**return leftNode;**

**}**

**struct nodeOfTree\* rotateleft(struct nodeOfTree\* node)**

**{**

**struct nodeOfTree\* rightNode = node->right;**

**node->right = rightNode->left;**

**rightNode->left = node;**

**fixHeight(node);**

**fixHeight(rightNode);**

**return rightNode;**

**}**

**struct nodeOfTree\* balance(struct nodeOfTree\* node) // балансировка узла p**

**{**

**fixHeight(node); // ведь +1 после рекурсии создания**

**if (bFactor(node) == 2)**

**{**

**if (bFactor(node->right) < 0)**

**node->right = rotateright(node->right);**

**return rotateleft(node);**

**}**

**if (bFactor(node) == -2)**

**{**

**if (bFactor(node->left) > 0)**

**node->left = rotateleft(node->left);**

**return rotateright(node);**

**}**

**return node; // балансировка не нужна**

**}**

**struct nodeOfTree\* insert(struct nodeOfTree\* root, char\* keySecondName)**

**{**

**if (!root)**

**{**

**struct nodeOfTree\* newNode = (struct nodeOfTree\*)calloc(1, sizeof(struct nodeOfTree));**

**strcpy(newNode->hash, keySecondName);**

**(newNode->repeats)++;**

**return newNode;**

**}**

**if (strcmp(keySecondName, root->hash) < 0)**

**root->left = insert(root->left, keySecondName);**

**if (strcmp(keySecondName, root->hash) > 0)**

**root->right = insert(root->right, keySecondName);**

**if (strcmp(keySecondName, root->hash) == 0) //Показать, что нет коллизий**

**{**

**(root->repeats)++;**

**return root;**

**}**

**return balance(root);**

**}**

**struct nodeOfTree\* findmin(struct nodeOfTree\* mainRoot)**

**{**

**return mainRoot->left ? findmin(mainRoot->left) : mainRoot;**

**}**

**struct nodeOfTree\* removemin(struct nodeOfTree\* node)**

**{**

**if (node->left == NULL)**

**return node->right;**

**node->left = removemin(node->left);**

**return balance(node);**

**}**

**struct nodeOfTree\* removeNode(struct nodeOfTree\* mainRoot, char\* removableHash)**

**{**

**if (!mainRoot) return NULL;**

**if (strcmp(removableHash, mainRoot->hash) < 0)**

**mainRoot->left = remove(mainRoot, removableHash);**

**else if (strcmp(removableHash, mainRoot->hash) > 0)**

**mainRoot->right = remove(mainRoot->right, removableHash);**

**else**

**{**

**struct nodeOfTree\* leftOfDelete = mainRoot->left;**

**struct nodeOfTree\* rightOfDelete = mainRoot->right;**

**free(mainRoot);**

**if (!rightOfDelete) return leftOfDelete;**

**struct nodeOfTree\* min = findmin(rightOfDelete);**

**min->right = removemin(rightOfDelete);**

**min->left = leftOfDelete;**

**return balance(min);**

**}**

**return balance(mainRoot);**

**}**

**void show\_Tree\_symmetrically(struct nodeOfTree\* root)**

**{**

**if (root)**

**{**

**if (root->left) show\_Tree\_symmetrically(root->left);**

**if (root->right) show\_Tree\_symmetrically(root->right);**

**printf("\nузел содержит : %s , число встреч %d\n", root->hash, root->repeats);**

**}**

**}**

**struct nodeOfTree\* cleanTree(struct nodeOfTree\* root)**

**{**

**if ((root))**

**{**

**if ((root)->left) cleanTree((root)->left);**

**if ((root)->right) cleanTree((root)->right);**

**free((root));**

**}**

**return NULL;**

**}**

**void CorrectInput(int\* Information)**

**{**

**int correct;**

**int flag = 0;**

**correct = scanf("%d", Information);**

**do**

**{**

**if (flag)**

**{**

**printf("Неверный ввод, попробуйте еще раз: ");**

**rewind(stdin);**

**correct = scanf("%d", Information);**

**}**

**flag = 1;**

**} while (!(correct) || (\*Information) <= 0);**

**}**

**struct nodeOfTree\* startWorkWithFile(struct nodeOfTree\* mainRoot)**

**{**

**char operation;**

**char\*\* accaunts;**

**if (!(accaunts = (char\*\*)calloc(1, sizeof(char\*))))**

**exit(1);**

**int count = 0;**

**printf("\nЕсли НЕ хотите загрузить аккаунты из файла, то нажмите 1");**

**printf("\nЕсли хотите загрузить аккаунты из файла, то нажмите 2\n");**

**rewind(stdin);**

**operation = getchar();**

**switch (operation)**

**{**

**case'1':**

**return mainRoot;**

**case'2':**

**accaunts=readFromFile(accaunts,&count);**

**printf("\nСодержимое файла:\n");**

**for (int i = 0; i < count; i++)**

**{**

**printf("%s\n", accaunts[i]);**

**mainRoot = insert(mainRoot, accaunts[i]);**

**}**

**return mainRoot;**

**}**

**}**

**char\*\* readFromFile(char\*\* accaunts, int\*count)**

**{**

**FILE\* file;**

**char fileName[maxLengthFile];**

**char currentHash[lengthHash+1];**

**re\_read:**

**printf("\nВведите название нужного файла: ");**

**rewind(stdin);**

**gets\_s(fileName, maxLengthFile);**

**if (!(file = fopen(fileName, "r")))**

**{**

**printf("\nОшибка, нужный файл не найден ");**

**goto re\_read;**

**}**

**rewind(file);**

**while (!feof(file))**

**{**

**while(1)**

**{**

**if (getc(file) != '\n')**

**if (feof(file))**

**return accaunts;**

**else**

**break;**

**}**

**fseek(file, -1, SEEK\_CUR);**

**(\*count)++;**

**if ( !( accaunts = (char\*\*)realloc(accaunts, (\*count)\*sizeof(char\*) ) ) )**

**exit(-1);**

**accaunts[(\*count) - 1] = (char\*)calloc(11, sizeof(char));**

**fgets(currentHash, lengthHash+1, file);**

**strcpy(accaunts[(\*count) - 1], currentHash);**

**}**

**return accaunts;**

**}**

**int NomatchPassword(struct nodeOfTree\* root, char\*hash)**

**{**

**int static check = 1;**

**int static count = 0;**

**if (root)**

**{**

**if (root->left)**

**{**

**count++;**

**NomatchPassword(root->left, hash);**

**}**

**if (root->right)**

**{**

**count++;**

**NomatchPassword(root->right, hash);**

**}**

**if (strcmp(hash, root->hash) == 0)**

**{**

**count--;**

**check = 0;**

**return check;**

**}**

**if (count == 0)**

**{**

**if (check == 0)**

**{**

**check = 1;// для будущих**

**return 0;**

**}**

**else**

**return 1;**

**}**

**count--;**

**}**

**return check;**

**}**

**char\*\* fromTreeToFile(struct nodeOfTree\* root)**

**{**

**FILE\* file;**

**char fileName[30];**

**re\_write:**

**printf("\nВведите название нового файла: ");**

**rewind(stdin);**

**gets\_s(fileName, 29);**

**if (!(file = fopen(fileName, "w")))**

**{**

**printf("\nОшибка, попробуйте еще раз");**

**goto re\_write;**

**}**

**rewind(file);**

**enumeration(root, file);**

**fclose(file);**

**}**

**void enumeration(struct nodeOfTree\* root, FILE\*file)**

**{**

**if (root)**

**{**

**if (root->left) enumeration(root->left, file);**

**if (root->right) enumeration(root->right, file);**

**fprintf(file, "%s", root->hash);**

**fprintf(file, "%c", '\n');**

**}**

**}**

**void inPutLoginAndPassword(char\*login, char\* password)**

**{**

**printf("\nВведите логин (не более 50 символов)");**

**rewind(stdin);**

**gets\_s(login, 50);**

**printf("\nВведите пароль (не более 50 символов)");**

**rewind(stdin);**

**gets\_s(password, 50);**

**}**

**char receivingExistAsciiCode(int x)**

**{**

**x += 122;**

**while (!(((x >= 48) && (x <= 57)) || ((x >= 65) && (x <= 90)) || ((x >= 97) && (x <= 122))))**

**x -= 46;**

**return x;**

**}**

**int getControlSum(char\*string)**

**{**

**int count = 0;**

**for (int i = 0; i < strlen(string); i++)**

**count += (string[i]);**

**return count;**

**}**

**char\* Hash(char\* userLogin,char\* userPassword, int lengthOfHash)**

**{**

**char codeSymbol[2];**

**codeSymbol[1] = '\0';**

**char\*userLoginAndPassword;**

**userLoginAndPassword= (char\*)calloc(strlen(userLogin)+strlen(userPassword)+1, sizeof(char));**

**strcat(userLoginAndPassword, userLogin);**

**strcat(userLoginAndPassword, userPassword);**

**char\* hash;**

**hash = (char\*)calloc(1, sizeof(char));**

**int minLen = 2;//Минимальная длина строки хеша, кратная двум**

**int realMinLen = 0;//Длина строки, ближайшая к нужной длине хеша**

**int originalSault = getControlSum(userLoginAndPassword);**

**int originalLenghtStr = strlen(userLoginAndPassword);**

**while (minLen <= lengthOfHash) realMinLen = (minLen \*= 2);**

**while (minLen < originalLenghtStr) minLen \*= 2;**

**minLen \*= 2; //Делаем длину строки хеша, как минимум, в 2 раза длинней оригинальной строки**

**int addCount = minLen - originalLenghtStr;**

**for (int i = 0; i < addCount; i++)**

**{**

**codeSymbol[0] = receivingExistAsciiCode(userLoginAndPassword[i] + userLoginAndPassword[i + 1]);**

**userLoginAndPassword = (char\*)realloc(userLoginAndPassword, strlen(userLoginAndPassword) + 2);**

**strcat(userLoginAndPassword, codeSymbol);**

**}**

**int maxSault = getControlSum(userLoginAndPassword);**

**int maxLenghStr = strlen(userLoginAndPassword);**

**while (strlen(userLoginAndPassword) != realMinLen)**

**{**

**for (int i = 0, center = strlen(userLoginAndPassword) / 2; i < center; i++)**

**{**

**codeSymbol[0] = receivingExistAsciiCode(userLoginAndPassword[center - i] + userLoginAndPassword[center + i]);**

**hash= (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**}**

**strcpy(userLoginAndPassword, hash);**

**hash = (char\*)realloc(hash, 1);**

**hash[0] = '\0';**

**}**

**int correction = realMinLen - lengthOfHash;**

**for (int i = 0, countCompress = realMinLen / correction; strlen(hash) < (lengthOfHash - 4); i++)**

**{**

**if (i%countCompress == 0)**

**{**

**codeSymbol[0] = receivingExistAsciiCode(userLoginAndPassword[i] + userLoginAndPassword[++i]);**

**hash = (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**}**

**else**

**{**

**codeSymbol[0] = userLoginAndPassword[i];**

**hash = (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**}**

**}**

**codeSymbol[0] = receivingExistAsciiCode(originalSault);**

**hash = (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**codeSymbol[0] = receivingExistAsciiCode(originalLenghtStr);**

**hash = (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**codeSymbol[0] = receivingExistAsciiCode(maxSault);**

**hash = (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**codeSymbol[0] = receivingExistAsciiCode(maxLenghStr);**

**hash = (char\*)realloc(hash, strlen(hash) + 2);**

**strcat(hash, codeSymbol);**

**return hash;**

**}**