Федеральное агентство связи  
Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
 «Сибирский государственный университет  
телекоммуникаций и информатики»

Кафедра вычислительных систем

Курсовая работа

по дисциплине «Технологии разработки программного обеспечения»

на тему «Password Generator»

Выполнил:

ст. гр. ИС-242

Макиенко В.С.

Проверил:

ст. преподаватель Токмашева Е. И.

**Содержание:**

[**Введение и постановка задачи 3**](#_Toc136728359)

[**Техническое задание 4**](#_Toc136728360)

[**Описание выполненного проекта 5**](#_Toc136728361)

[**Считывания данных 6**](#_Toc136728362)

[**Выборка нужных символов 6**](#_Toc136728363)

[**Генерация символа для пароля 6**](#_Toc136728364)

[**Генерация пароля 6**](#_Toc136728365)

[**Вывод пароля 6**](#_Toc136728366)

[**Личный вклад в проект 7**](#_Toc136728367)

[**Приложение. Текст программы 12**](#_Toc136728368)

# Введение и постановка задачи

Основная наша задача была в закреплении знаний полученных на предмете ТРПО. В качестве курсовой работы мы решили работать над проектом Password Generator. Пред нами стоял такой список задач для осуществления этого проекта.

* Добавить файл с функциями (это основной файл в котором и будут генерироваться пароли)
* Добавить файл для вызова функций
* Добавить библиотеку (В ней будет храниться структура для работы функций а так же она будет связывать файлы между собой)
* Добавить Makefile (этот файл нужен для упрощения компиляции программы)
* Добавить CI ( Для проверки работоспособности приложения)
* Добавить папку с файлом, в который будут записываться пороли
* Покрыть файл с функциями тестами

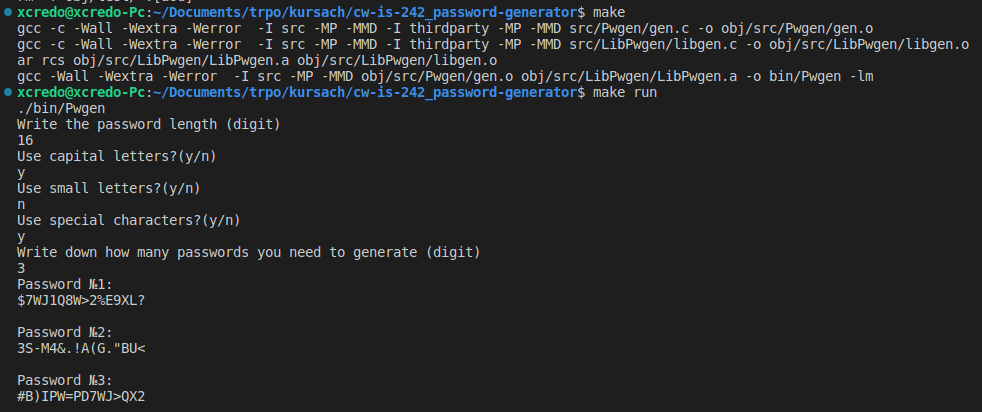
# Техническое задание

1. Функциональность проекта: Генератор паролей. Предполагает использование в различных сферах однофакторной аутентификации, посредством генерации пароля из различных символов и регистров;
2. Приложение должно предоставлять пользователю возможность выбора длины пароля, использования заглавных и строчных букв, а также специальных символов;
3. Пользователь должен иметь возможность указать количество паролей, которые необходимо сгенерировать;
4. Генерация паролей должна происходить посредством использования функций, описанных в библиотеках;
5. Все полученные пароли должны отображаться на экране в виде текста;
6. Приложение должно проверять пользовательский ввод на корректность;
7. Приложение должно предоставлять возможность сохранения результатов в файл.

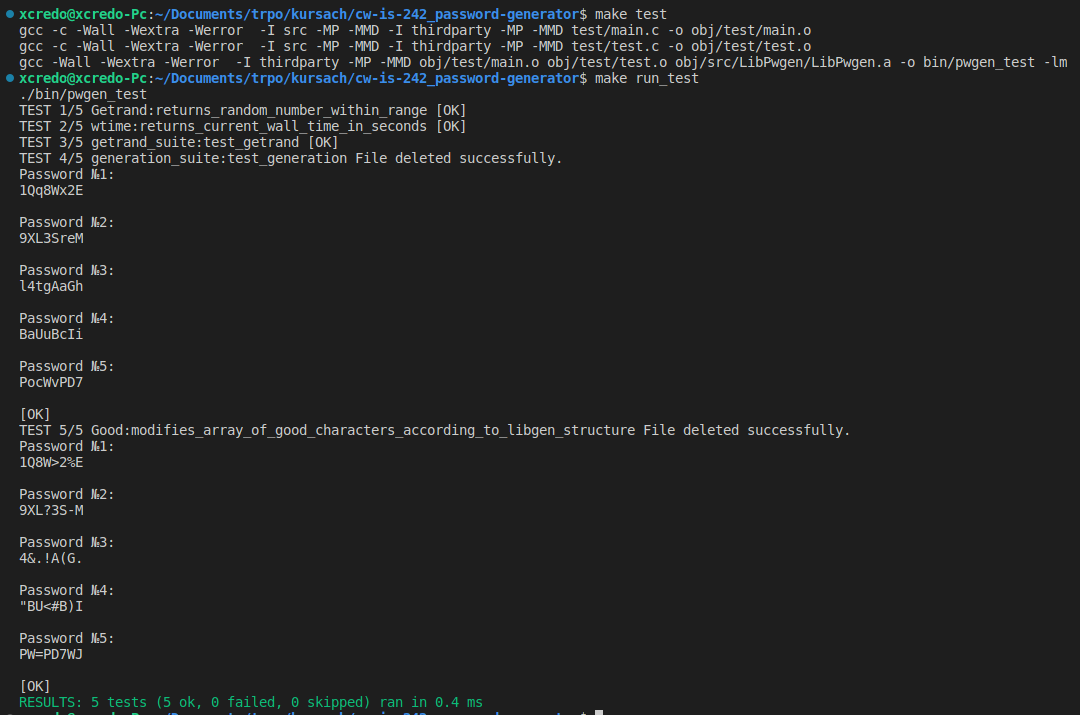
# Описание выполненного проекта

Общекомандная часть с примерами работы ПО.

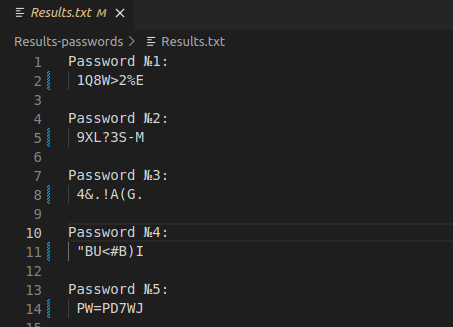
**Функционал работы кода:**



Работа генератора паролей.



Работа тестов функций.



Запись результатов работы кода в текстовый файл.

**Описание наименований функций:**

## Считывания данных

Ui – Функция которая создает интерфейс и запрашивает у пользователя какие символы использовать в генерации пароля.

## Выборка нужных символов

На основе данных введенными пользователем функция Good оставляет символы, которые будут использоваться в пароле.

## Генерация символа для пароля

Getrand – функция генерирующая произвольное число в нужном нам диапазоне и передающая его в функцию Genetation

## Генерация пароля

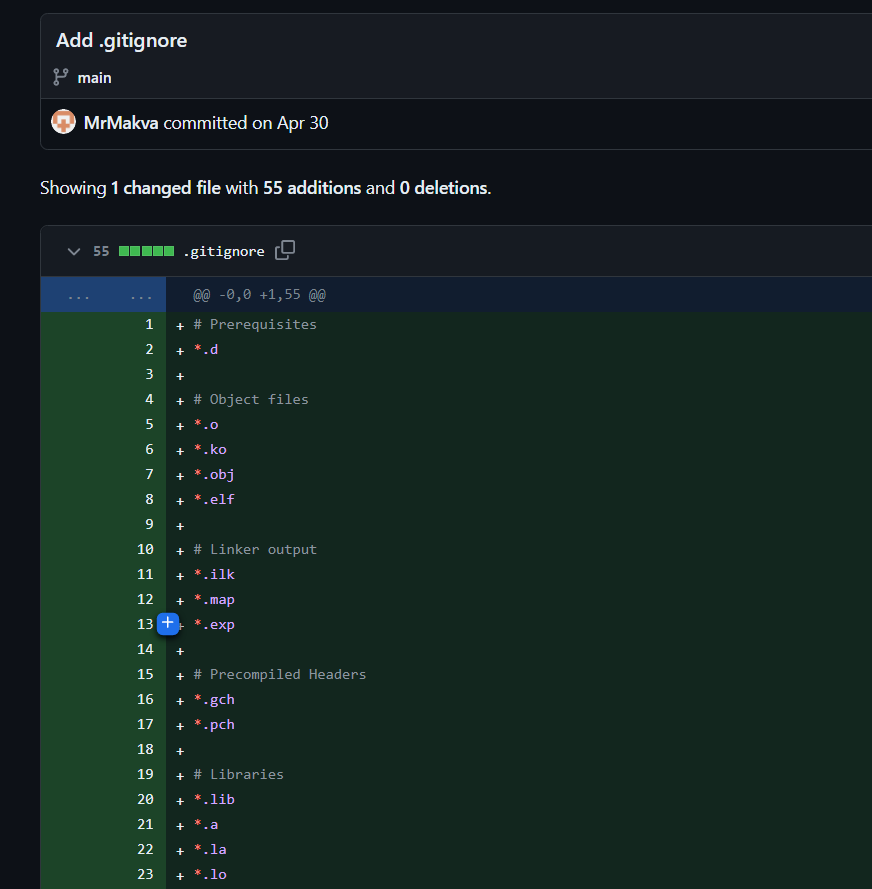
Generation –Получает число на вход, с помощью таблице ACII переводит его в символ и проверяет проходит ли оно условия введенные пользователем.

## Вывод пароля

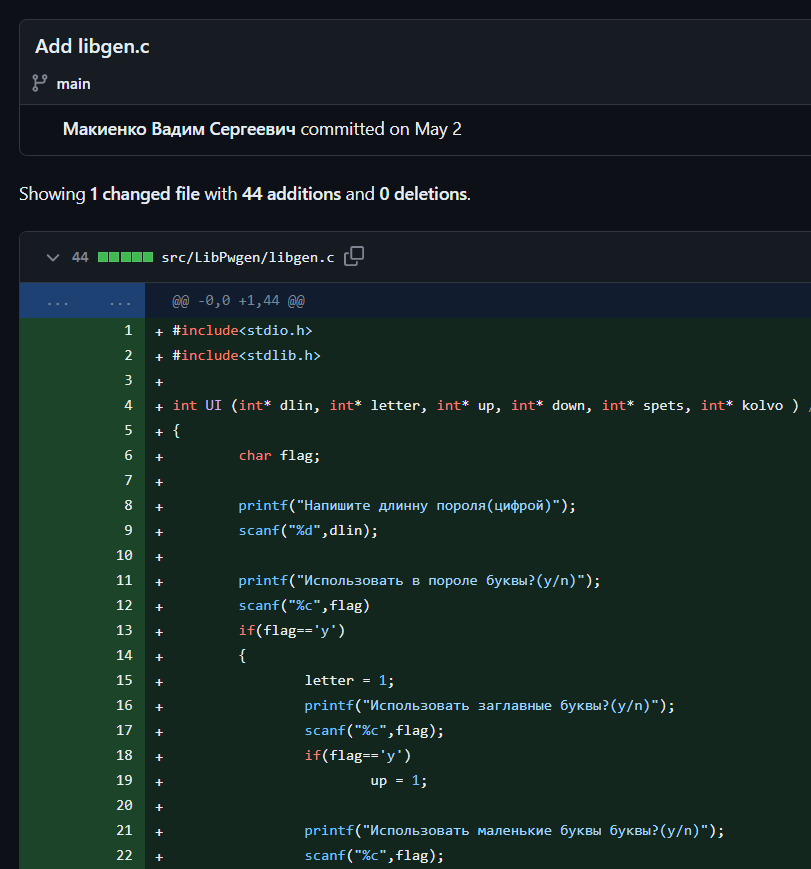
Output – функция вывода пароля в терминал а также записи его в файл Results.txt

# Личный вклад в проект

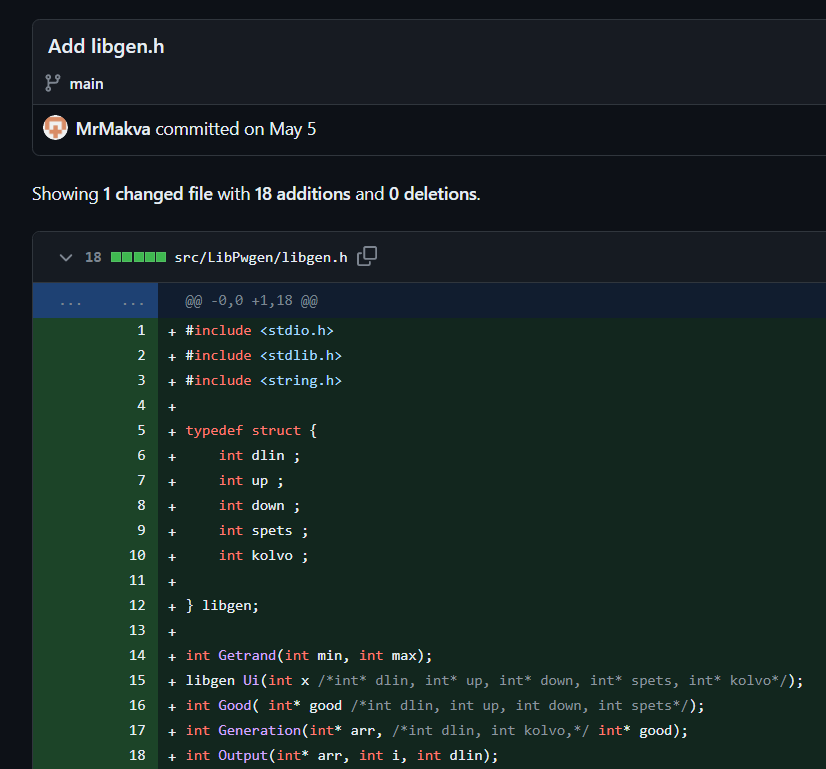
* + Добавил gitignore



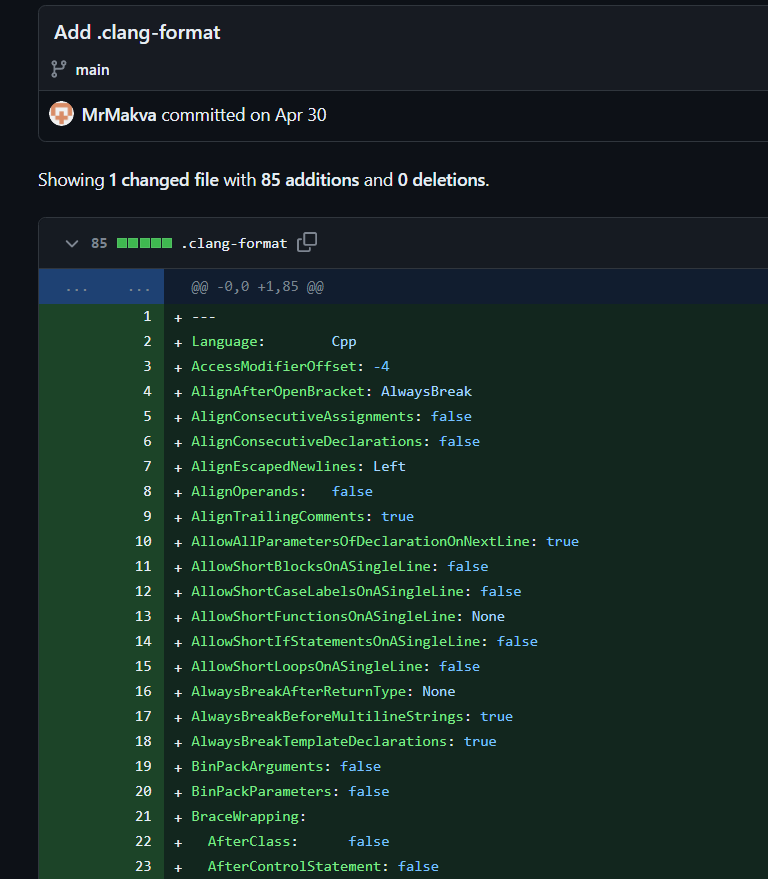
* + Создал файл для функций генерации паролей и написал их



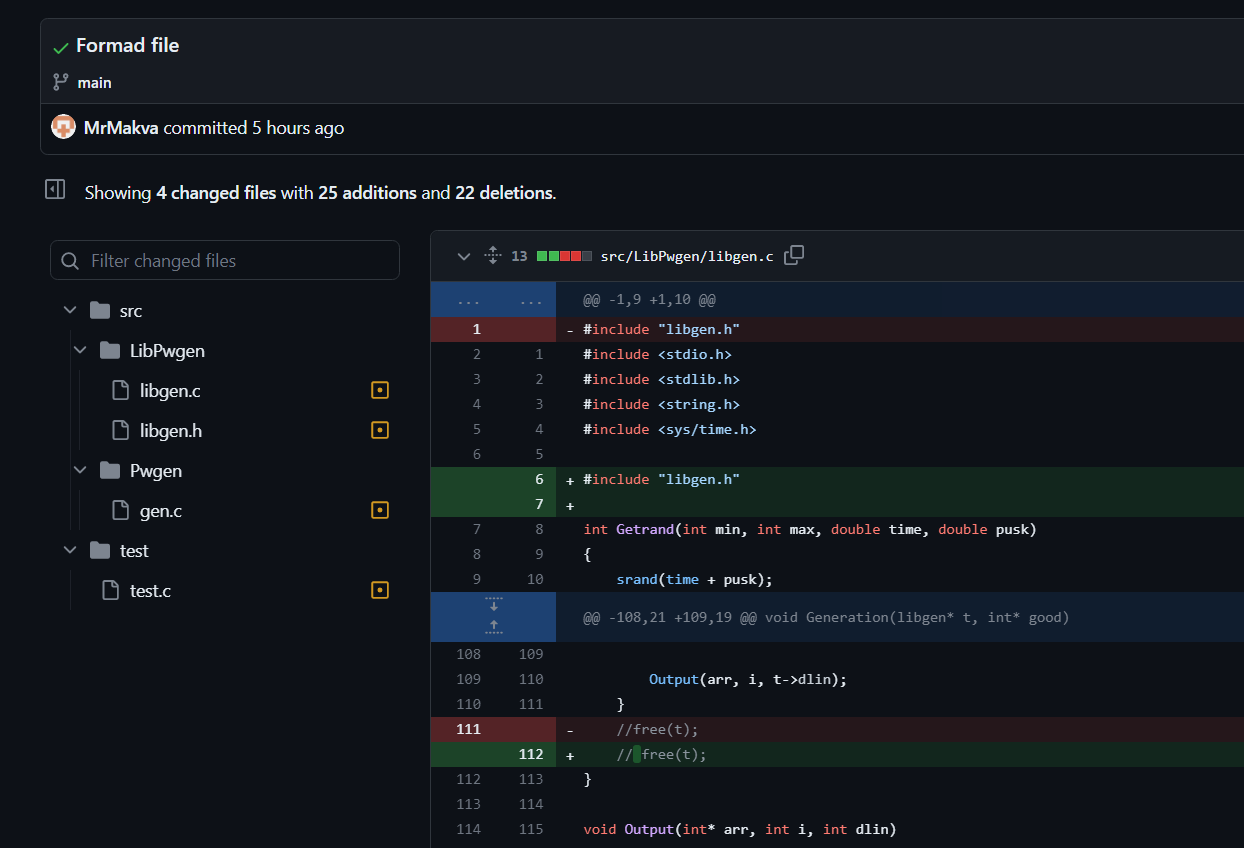
* + Создал заголовочный файл, которой будет связывать остальные файлы



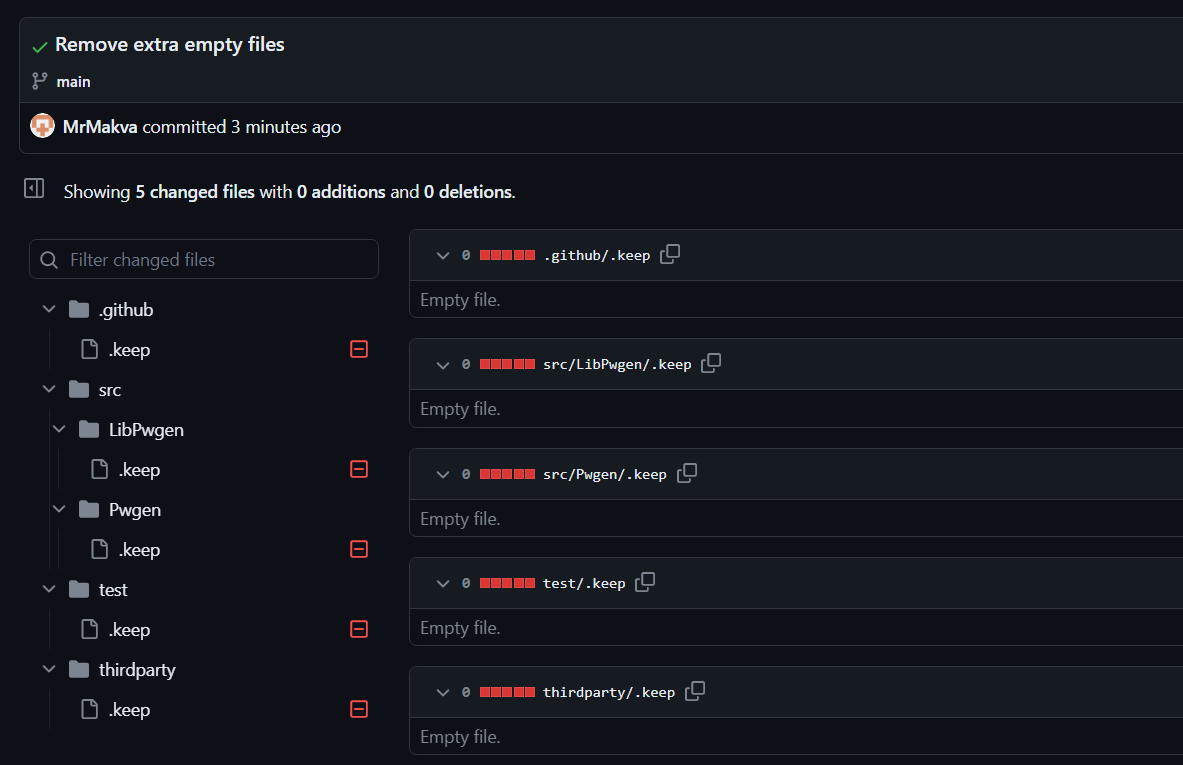
* + Добавил и настроил clang-format



* + Отформатировал файлы и исправил ошибки на которые указывал CI



* + Удалил лишние пустые файлы (нужны были в самом начале для создания папок)



# Приложение. Текст программы

**// file gen.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include <stdio.h>  #include "../LibPwgen/libgen.h"  int main()  {  libgen\* t = malloc(**sizeof**(libgen));  Ui(t);  **return** 0;  } |

**// file libgen.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129 | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/time.h>  #include "libgen.h"  int Getrand(int min, int max, double time, double pusk)  {  srand(time + pusk);  **return** (double)rand() / (RAND\_MAX + 1.0) \* (max - min) + min;  }  double wtime()  {  **struct** timeval t;  gettimeofday(&t, NULL);  **return** (double)t.tv\_sec + (double)t.tv\_usec \* 1E-6;  }  void Ui(libgen\* t)  {  t = malloc(**sizeof**(libgen) \* t->kolvo);  char flag;  printf("Write the password length (digit)**\n**");  scanf(" %d", &t->dlin);  printf("Use capital letters?(y/n)**\n**");  scanf(" %c", &flag);  **if** (flag == 'y')  t->up = 1;  printf("Use small letters?(y/n)**\n**");  scanf(" %c", &flag);  **if** (flag == 'y')  t->down = 1;  printf("Use special characters?(y/n)**\n**");  scanf(" %c", &flag);  **if** (flag == 'y')  t->spets = 1;  printf("Write down how many passwords you need to generate (digit)**\n**");  scanf(" %d", &t->kolvo);  Good(t);  }  void Good(libgen\* t)  {  int good[123];  **for** (int i = 33; i < 122; i++) {  good[i] = 1;  }  **for** (int i = 91; i <= 96; i++) {  good[i] = 0;  }  **if** (t->up == 0) {  **for** (int i = 65; i <= 90; i++) {  good[i] = 0;  }  }  **if** (t->down == 0) {  **for** (int i = 97; i <= 122; i++) {  good[i] = 0;  }  }  **if** (t->spets == 0) {  **for** (int i = 33; i <= 47; i++) {  good[i] = 0;  }  **for** (int i = 58; i <= 64; i++) {  good[i] = 0;  }  }  Generation(t, good);  }  void Generation(libgen\* t, int\* good)  {  double time = 0;  double pusk = 0;  int arr[t->dlin];  int i = 0;  int min = 33;  int max = 122;  time = wtime();  int tmp = Getrand(min, max, time, pusk);  pusk += 1;  **for** (i = 1; i < t->kolvo + 1; i++) {  **for** (int j = 0; j < t->dlin; j++) {  **while** (good[tmp] == 0) {  time = wtime();  tmp = Getrand(min, max, time, pusk);  pusk += 1;  }  arr[j] = tmp;  time = wtime();  tmp = Getrand(min, max, time, pusk);  pusk += 1;  }  Output(arr, i, t->dlin);  }  *// free(t);*  }  void Output(int\* arr, int i, int dlin)  {  FILE\* file = fopen("Results-passwords/Results.txt", "a");  fprintf(file, "Password №%d:**\n** ", i);  printf("Password №%d:**\n**", i);  **for** (int x = 0; x < dlin; x++) {  printf("%c", arr[x]);  fprintf(file, "%c", arr[x]);  }  **for** (int rev = 0; rev < 2; rev++) {  printf("**\n**");  fprintf(file, "**\n**");  }  fclose(file);  } |

**// file libgen.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  **typedef** **struct** {  int dlin;  int up;  int down;  int spets;  int kolvo;  } libgen;  int Getrand(int min, int max, double time, double pusk);  void Ui(libgen\* t);  void Good(libgen\* t);  void Generation(libgen\* t, int\* good);  void Output(int\* arr, int i, int dlin);  double wtime(); |

**// file main.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | #define CTEST\_MAIN  #include <ctest.h>  int main(int argc, **const** char\*\* argv)  {  **return** ctest\_main(argc, argv);  } |

**// file test.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115 | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/time.h>  #include "../src/LibPwgen/libgen.h"  #include "../thirdparty/ctest.h"  int remfile()  {  **if** (remove("Results-passwords/Results.txt") == 0) {  printf("File deleted successfully.**\n**");  } **else** {  printf("Failed to delete the file.**\n**");  }  **return** 0;  }  CTEST(Getrand, returns\_random\_number\_within\_range)  {  int min = 0;  int max = 10;  double time = 12345.0;  double pusk = 6789.0;  int result = Getrand(min, max, time, pusk);  ASSERT\_TRUE(result >= min && result <= max);  }  CTEST(wtime, returns\_current\_wall\_time\_in\_seconds)  {  double result1 = wtime();  double result2 = wtime();  ASSERT\_TRUE(result1 <= result2);  }  CTEST(getrand\_suite, test\_getrand)  {  int rand\_num = Getrand(1, 10, 0.5, 0.2);  ASSERT\_GE(rand\_num, 1);  ASSERT\_LE(rand\_num, 10);  }  CTEST(generation\_suite, test\_generation)  {  remfile();  *// Create the libgen structure*  libgen\* t = malloc(**sizeof**(libgen));  t->dlin = 8;  t->up = 1;  t->down = 1;  t->spets = 0;  t->kolvo = 5;  *// Create an array good and fill it*  int good[123] = {0};  **for** (int i = 48; i <= 57; i++) {  good[i] = 1; *// numbers from 0 to 9*  }  **for** (int i = 65; i <= 90; i++) {  good[i] = 1; *// capital letters*  }  **for** (int i = 97; i <= 122; i++) {  good[i] = 1; *// lower case*  }  *// Execute the Generation function*  Generation(t, good);  *// Checking for the existence of a file with the results of generating*  *// passwords*  FILE\* file = fopen("Results-passwords/Results.txt", "r+");  ASSERT\_NOT\_NULL(file);  *// Checking the number of generated passwords*  char buffer[1024];  int cnt = 0;  **while** (fgets(buffer, 1024, file) != NULL) {  **if** (strstr(buffer, "Password")) {  cnt++;  }  }  ASSERT\_EQUAL(cnt, t->kolvo);  fclose(file);  free(t);  }  CTEST(Good, modifies\_array\_of\_good\_characters\_according\_to\_libgen\_structure)  {  remfile();  libgen t = {8, 1, 0, 1, 5};  int good[123];  **for** (int i = 33; i <= 122; i++) {  good[i] = 1;  }  Good(&t);  ASSERT\_TRUE(good['a']);  ASSERT\_TRUE(good['!']);  ASSERT\_TRUE(good['@']);  ASSERT\_TRUE(good['#']);  ASSERT\_TRUE(good['$']);  ASSERT\_TRUE(good['%']);  ASSERT\_TRUE(good['^']);  ASSERT\_TRUE(good['&']);  ASSERT\_TRUE(good['\*']);  *// Other characters must not be changed*  ASSERT\_TRUE(good['B']);  ASSERT\_TRUE(good['z']);  ASSERT\_TRUE(good['~']);  } |