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
Журбенко В.Е.

Проверил:
ст. преподаватель Токмашева Е. И.

Новосибирск, 03.06.2023

Содержание:

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

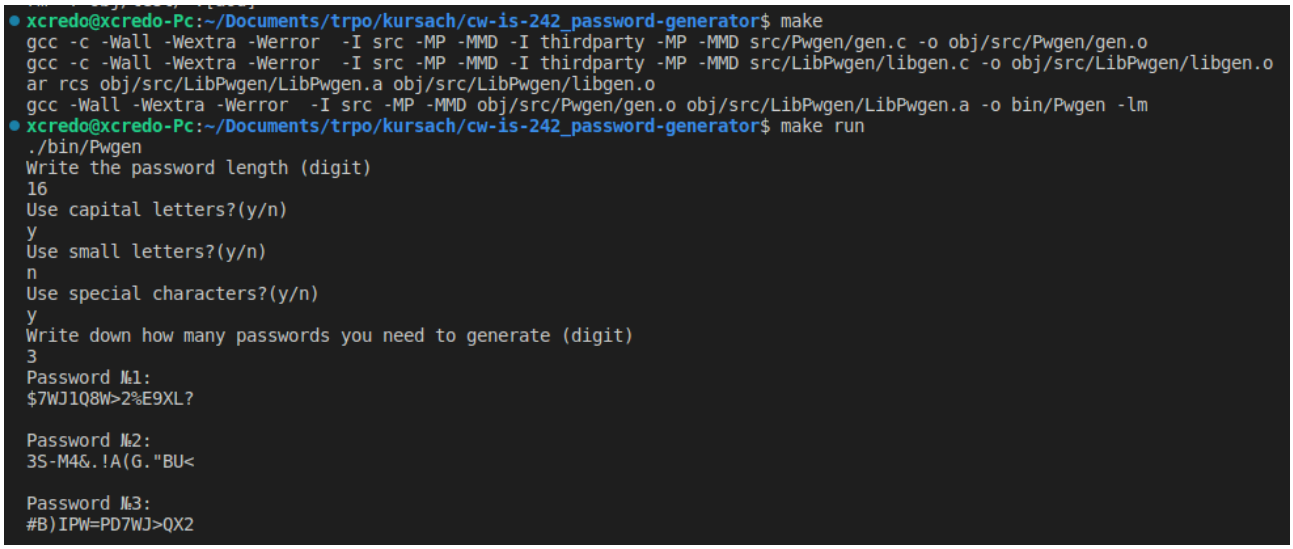
Техническое задание

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.

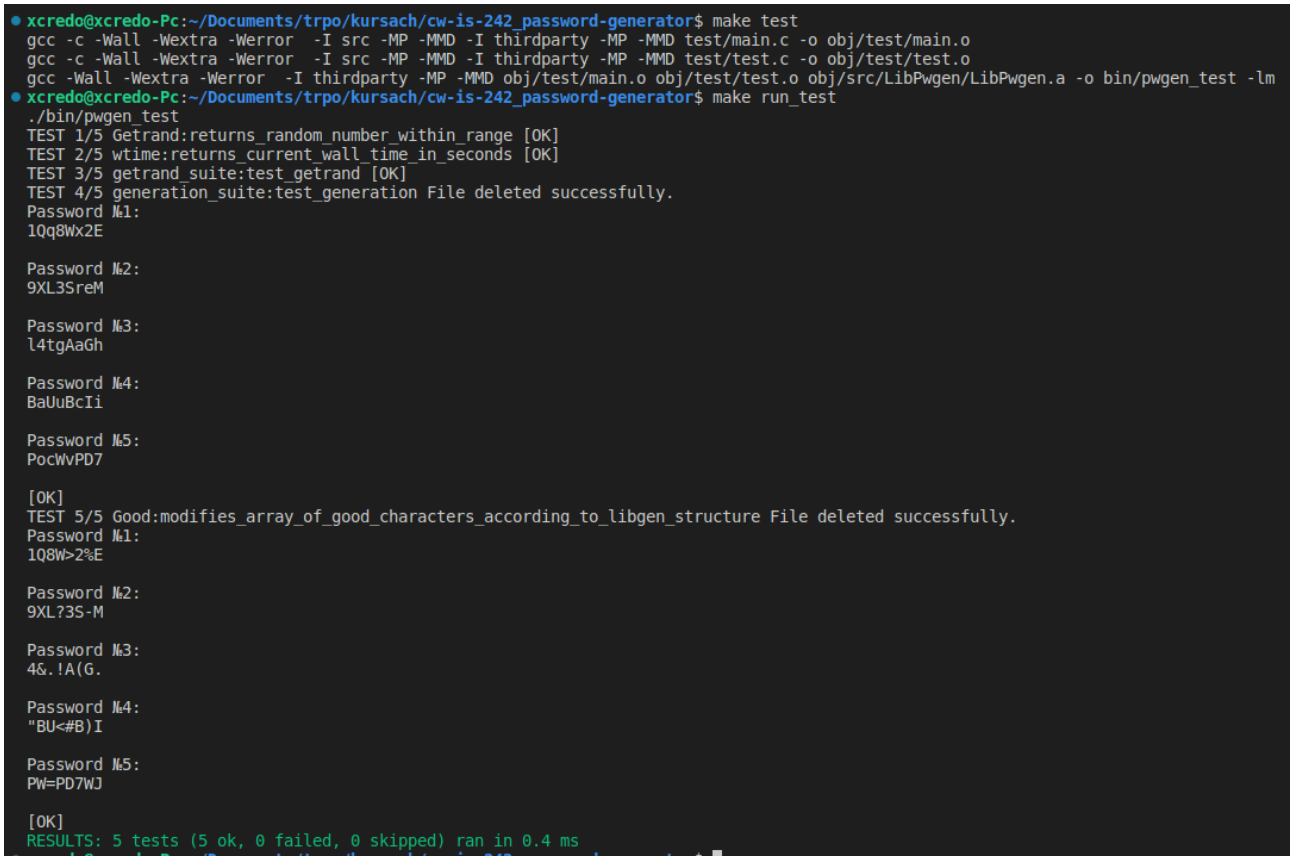
Описание выполненного проекта

Общекомандная часть с примерами работы ПО.

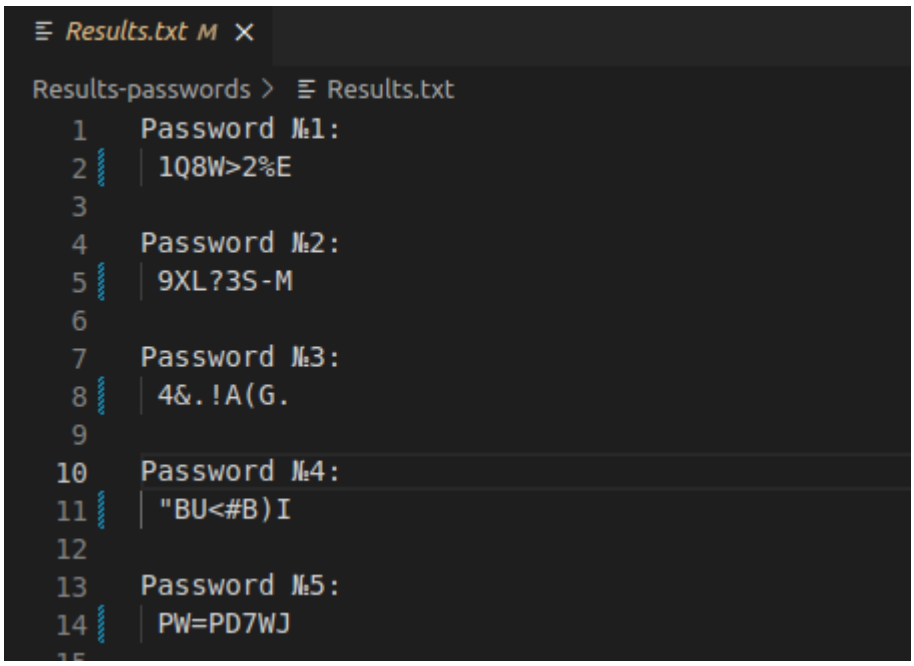
Функционал работы кода:



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

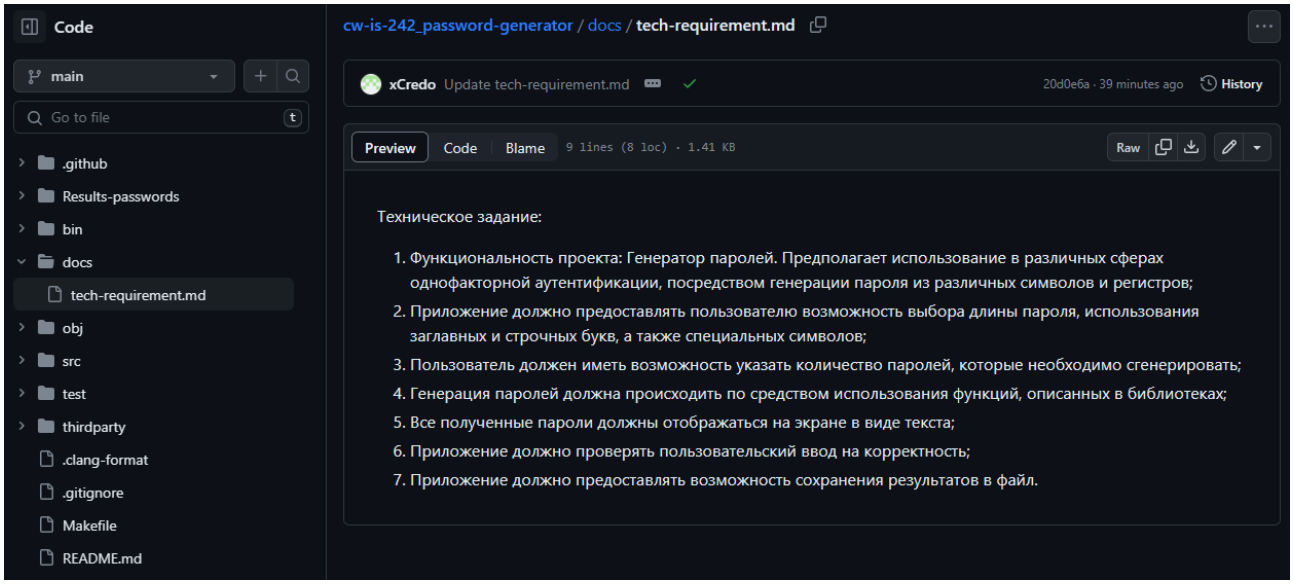
Generation –Получает число на вход, с помощью таблице ACII переводит его в символ и проверяет проходит ли оно условия введенные пользователем.

Вывод пароля

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

Личный вклад в проект

1. Разработал ТЗ проекта;



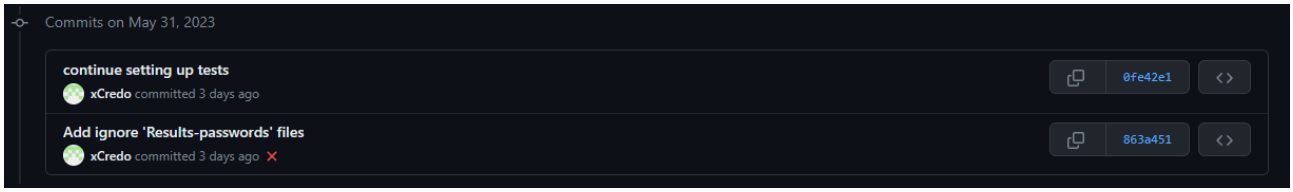
2. Разработал gen.c для взаимодействия со всеми функциями;

```c
#include <stdio.h>
#include "../LibPwgen/libgen.h"

int main()
{
    libgen* t = malloc(sizeof(libgen));
    Ui(t);
    return 0;
}
```

3. Внёс вклад в настройку libgen.c, .gitignore, Makefile;

Добавил возможность сохранения результатов в текстовый файл (Без перезаписи)



Добавил в .gitignore папку ‘Results-passwords’, в которой сохраняются сгенерированные пароли

4. Покрыл тестами функциональную часть программы.

```c
#define CTEST_MAIN

#include <ctest.h>

int main(int argc, const char** argv)
{
    return ctest_main(argc, argv);
}
```

Main.c для взаимодействия с тестами

```c
#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['~']);
}
```

Test.c для покрытия функций тестами

Приложение. Текст программы

// file gen.c

```c
#include <stdio.h>
#include "../LibPwgen/libgen.h"

int main()
{
    libgen* t = malloc(sizeof(libgen));
    Ui(t);
    return 0;
}
```

// file libgen.c

```c
#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;
```

```
43
44      printf("Write down how many passwords you need to generate (digit)\n");
45      scanf(" %d", &t->kolvo);
46      Good(t);
47  }
48
49  void Good(libgen* t)
50  {
51      int good[123];
52
53      for (int i = 33; i < 122; i++) {
54          good[i] = 1;
55      }
56
57      for (int i = 91; i <= 96; i++) {
58          good[i] = 0;
59      }
60
61      if (t->up == 0) {
62          for (int i = 65; i <= 90; i++) {
63              good[i] = 0;
64          }
65      }
66
67      if (t->down == 0) {
68          for (int i = 97; i <= 122; i++) {
69              good[i] = 0;
70          }
71      }
72
73      if (t->spets == 0) {
74          for (int i = 33; i <= 47; i++) {
75              good[i] = 0;
76          }
77
78          for (int i = 58; i <= 64; i++) {
79              good[i] = 0;
80          }
81      }
82      Generation(t, good);
83  }
84
85  void Generation(libgen* t, int* good)
86  {
87      double time = 0;
88      double pusk = 0;
89      int arr[t->dlin];
90      int i = 0;
91      int min = 33;
92      int max = 122;
93      time = wtime();
94      int tmp = Getrand(min, max, time, pusk);
95      pusk += 1;
96
97      for (i = 1; i < t->kolvo + 1; i++) {
98          for (int j = 0; j < t->dlin; j++) {
99              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

```
#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

```
#define CTEST_MAIN

#include <ctest.h>
```

```
4
5 int main(int argc, const char** argv)
6 {
7     return ctest_main(argc, argv);
8 }
```

// file test.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include <sys/time.h>
 5
 6 #include "../src/LibPwgen/libgen.h"
 7 #include "../thirdparty/ctest.h"
 8
 9 int remfile()
10 {
11     if (remove("Results-passwords/Results.txt") == 0) {
12         printf("File deleted successfully.\n");
13     } else {
14         printf("Failed to delete the file.\n");
15     }
16     return 0;
17 }
18
19 CTEST(Getrand, returns_random_number_within_range)
20 {
21     int min = 0;
22     int max = 10;
23     double time = 12345.0;
24     double pusk = 6789.0;
25     int result = Getrand(min, max, time, pusk);
26
27     ASSERT_TRUE(result >= min && result <= max);
28 }
29
30 CTEST(wtime, returns_current_wall_time_in_seconds)
31 {
32     double result1 = wtime();
33     double result2 = wtime();
34
35     ASSERT_TRUE(result1 <= result2);
36 }
37
38 CTEST(getrand_suite, test_getrand)
39 {
40     int rand_num = Getrand(1, 10, 0.5, 0.2);
41     ASSERT_GE(rand_num, 1);
42     ASSERT_LE(rand_num, 10);
43 }
44
45 CTEST(generation_suite, test_generation)
46 {
47     remfile();
48     // Create the libgen structure
49     libgen* t = malloc(sizeof(libgen));
```

```
50      t->dlin = 8;
51      t->up = 1;
52      t->down = 1;
53      t->spets = 0;
54      t->kolvo = 5;
55
56      // Create an array good and fill it
57      int good[123] = {0};
58      for (int i = 48; i <= 57; i++) {
59          good[i] = 1; // numbers from 0 to 9
60      }
61      for (int i = 65; i <= 90; i++) {
62          good[i] = 1; // capital letters
63      }
64      for (int i = 97; i <= 122; i++) {
65          good[i] = 1; // lower case
66      }
67
68      // Execute the Generation function
69      Generation(t, good);
70
71      // Checking for the existence of a file with the results of generating
72      // passwords
73      FILE* file = fopen("Results-passwords/Results.txt", "r+");
74      ASSERT_NOT_NULL(file);
75
76      // Checking the number of generated passwords
77      char buffer[1024];
78      int cnt = 0;
79      while (fgets(buffer, 1024, file) != NULL) {
80          if (strstr(buffer, "Password")) {
81              cnt++;
82          }
83      }
84      ASSERT_EQUAL(cnt, t->kolvo);
85
86      fclose(file);
87      free(t);
88  }
89
90  CTEST(Good, modifies_array_of_good_characters_according_to_libgen_structure)
91  {
92      remfile();
93      libgen t = {8, 1, 0, 1, 5};
94
95      int good[123];
96      for (int i = 33; i <= 122; i++) {
97          good[i] = 1;
98      }
99
100     Good(&t);
101     ASSERT_TRUE(good['a']);
102     ASSERT_TRUE(good['!']);
103     ASSERT_TRUE(good['@']);
104     ASSERT_TRUE(good['#']);
105     ASSERT_TRUE(good['$']);
106     ASSERT_TRUE(good['%']);
```

```
107     ASSERT_TRUE(good['^']);
108     ASSERT_TRUE(good['&']);
109     ASSERT_TRUE(good['*']);
110
111     // Other characters must not be changed
112     ASSERT_TRUE(good['B']);
113     ASSERT_TRUE(good['z']);
114     ASSERT_TRUE(good['~']);
115 }
```