МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра телекоммуникационных систем и вычислительных средств

(ТС и ВС)

# ОТЧЁТ

по дисциплине

«Программирование»

по теме:Создание приложения для тренировки печати.

Студент:

ИКС-431 Таубин Георгий

Преподователь:

А.И. Вейлер

Новосибирск 2025

## СОДЕРЖАНИЕ

1. Введение
2. Основные функции программы
3. Структура проекта
4. Скриншоты работы проекта
5. Тестирование
6. Заключение
7. Листринг кода

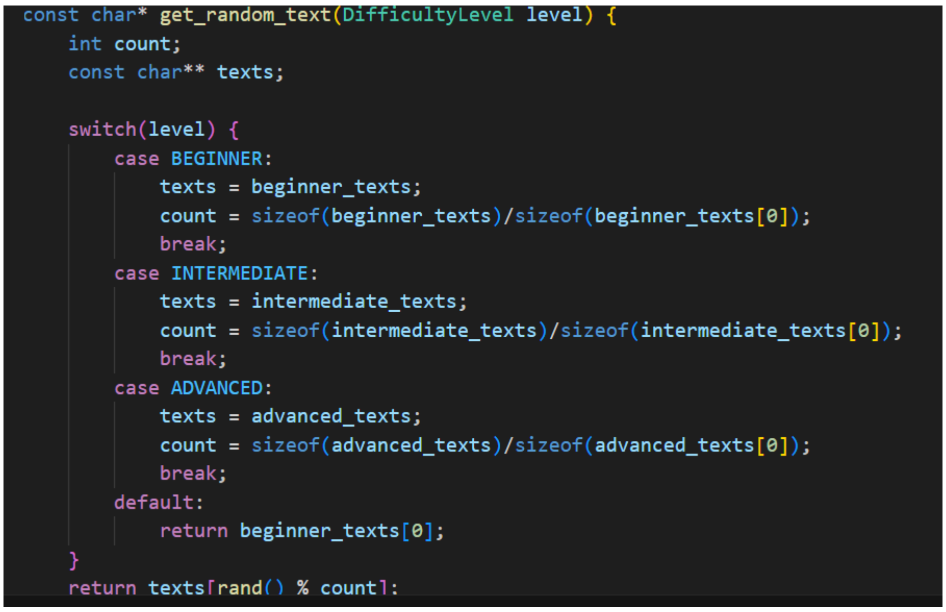
## 1. Введение

Целью данной работы являлась разработка консольного приложения для тренировки скорости и точности печати на клавиатуре. Приложение включает несколько режимов работы, статистику, таблицу рекордов и поддержку многопоточности.

## 2. Основные функции приложения

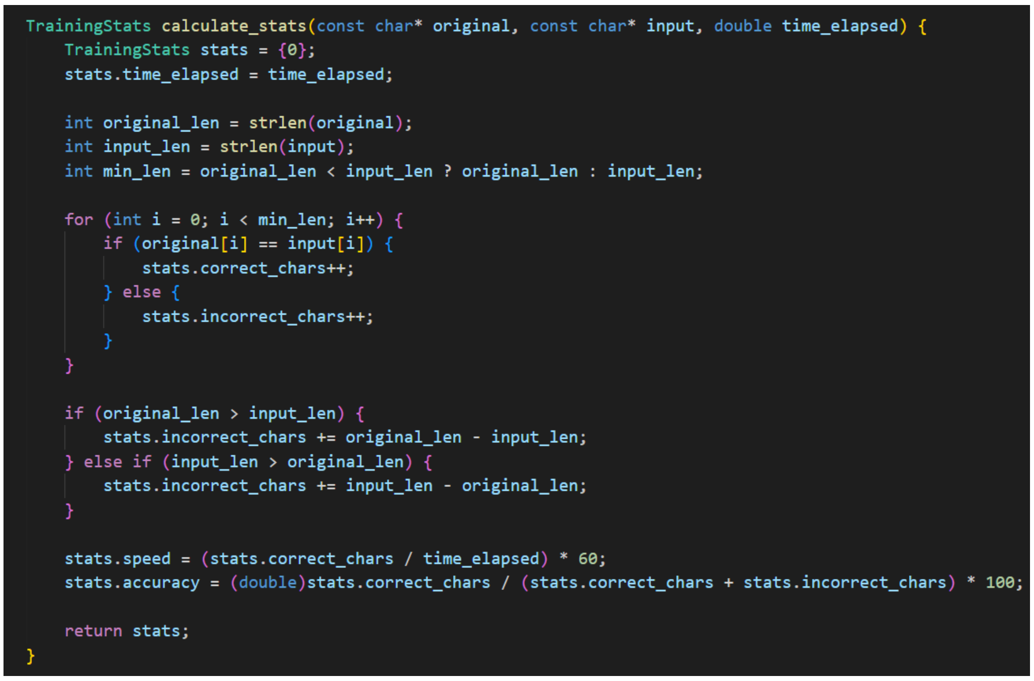
### 2.1. Режимы тренировки

1. Стандартный режим
    - Пользователь получает текст для набора.



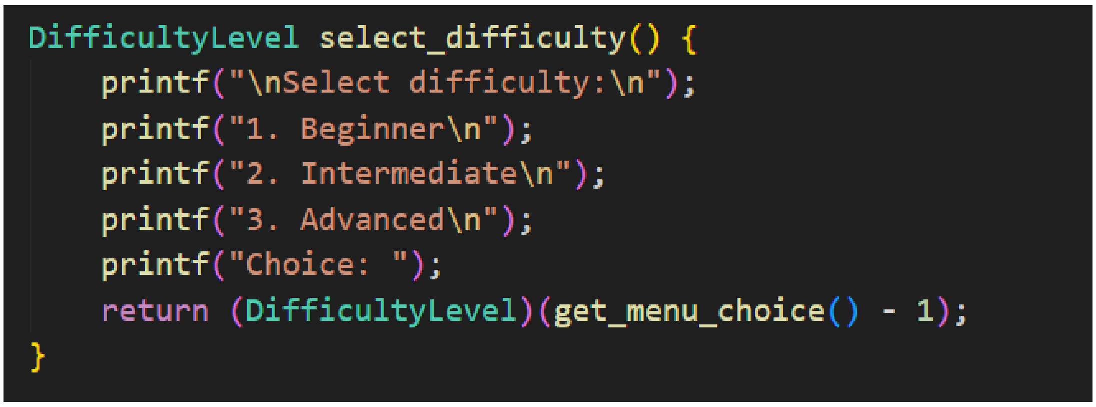
```
const char* get_random_text(DifficultyLevel level) {
    int count;
    const char** texts;

    switch(level) {
        case BEGINNER:
            texts = beginner_texts;
            count = sizeof(beginner_texts)/sizeof(beginner_texts[0]);
            break;
        case INTERMEDIATE:
            texts = intermediate_texts;
            count = sizeof(intermediate_texts)/sizeof(intermediate_texts[0]);
            break;
        case ADVANCED:
            texts = advanced_texts;
            count = sizeof(advanced_texts)/sizeof(advanced_texts[0]);
            break;
        default:
            return beginner_texts[0];
    }
    return texts[rand() % count];
```

    - После ввода выводятся статистика: скорость, точность, количество ошибок.



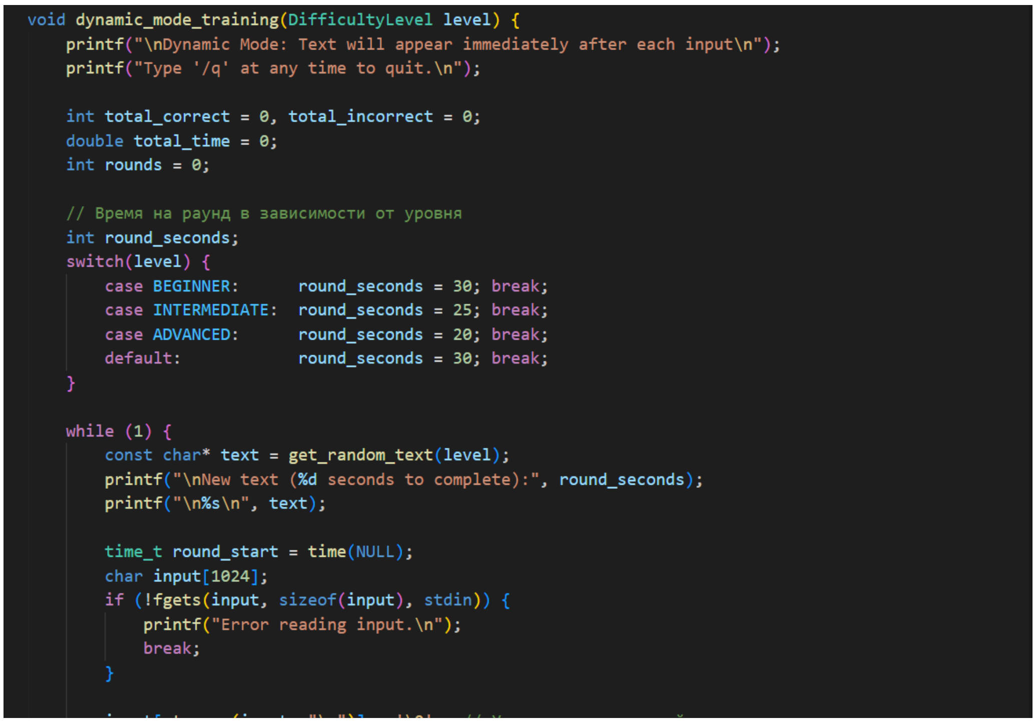
```c
TrainingStats calculate_stats(const char* original, const char* input, double time_elapsed) {
    TrainingStats stats = {0};
    stats.time_elapsed = time_elapsed;

    int original_len = strlen(original);
    int input_len = strlen(input);
    int min_len = original_len < input_len ? original_len : input_len;

    for (int i = 0; i < min_len; i++) {
        if (original[i] == input[i]) {
            stats.correct_chars++;
        } else {
            stats.incorrect_chars++;
        }
    }

    if (original_len > input_len) {
        stats.incorrect_chars += original_len - input_len;
    } else if (input_len > original_len) {
        stats.incorrect_chars += input_len - original_len;
    }

    stats.speed = (stats.correct_chars / time_elapsed) * 60;
    stats.accuracy = (double)stats.correct_chars / (stats.correct_chars + stats.incorrect_chars) * 100;

    return stats;
}
```

- Реализована поддержка разных уровней сложности (Beginner, Intermediate, Advanced).



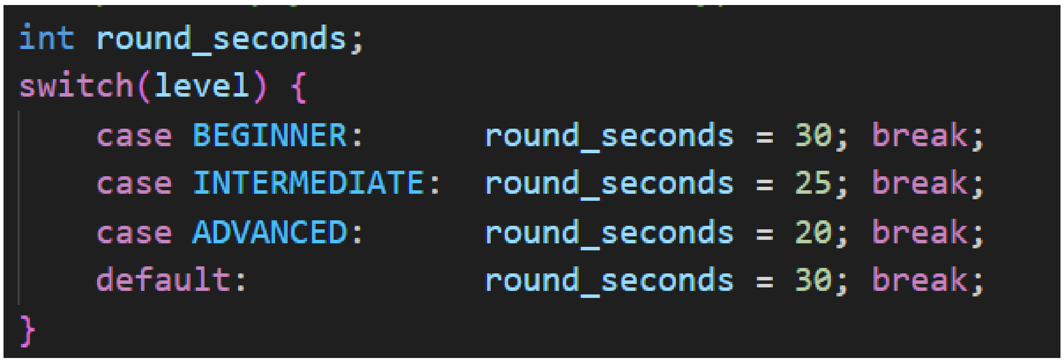
```c
DifficultyLevel select_difficulty() {
    printf("\nSelect difficulty:\n");
    printf("1. Beginner\n");
    printf("2. Intermediate\n");
    printf("3. Advanced\n");
    printf("Choice: ");
    return (DifficultyLevel)(get_menu_choice() - 1);
}
```

2. Динамический режим
   - Текст меняется после каждого ввода.



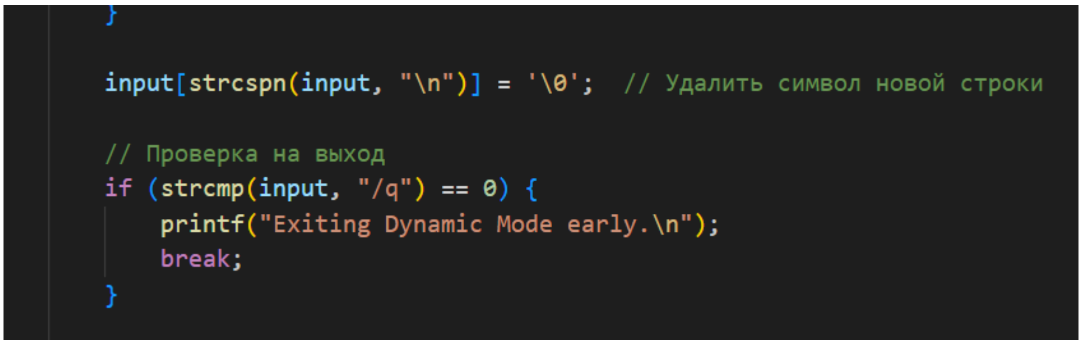
```c
void dynamic_mode_training(DifficultyLevel level) {
    printf("\nDynamic Mode: Text will appear immediately after each input\n");
    printf("Type '/q' at any time to quit.\n");

    int total_correct = 0, total_incorrect = 0;
    double total_time = 0;
    int rounds = 0;

    // Время на раунд в зависимости от уровня
    int round_seconds;
    switch(level) {
        case BEGINNER:      round_seconds = 30; break;
        case INTERMEDIATE:  round_seconds = 25; break;
        case ADVANCED:      round_seconds = 20; break;
        default:            round_seconds = 30; break;
    }

    while (1) {
        const char* text = get_random_text(level);
        printf("\nNew text (%d seconds to complete):", round_seconds);
        printf("\n%s\n", text);

        time_t round_start = time(NULL);
        char input[1024];
        if (!fgets(input, sizeof(input), stdin)) {
            printf("Error reading input.\n");
            break;
        }
```

- Ограничение по времени на ввод.



```c
int round_seconds;
switch(level) {
    case BEGINNER:      round_seconds = 30; break;
    case INTERMEDIATE:  round_seconds = 25; break;
    case ADVANCED:      round_seconds = 20; break;
    default:            round_seconds = 30; break;
}
```

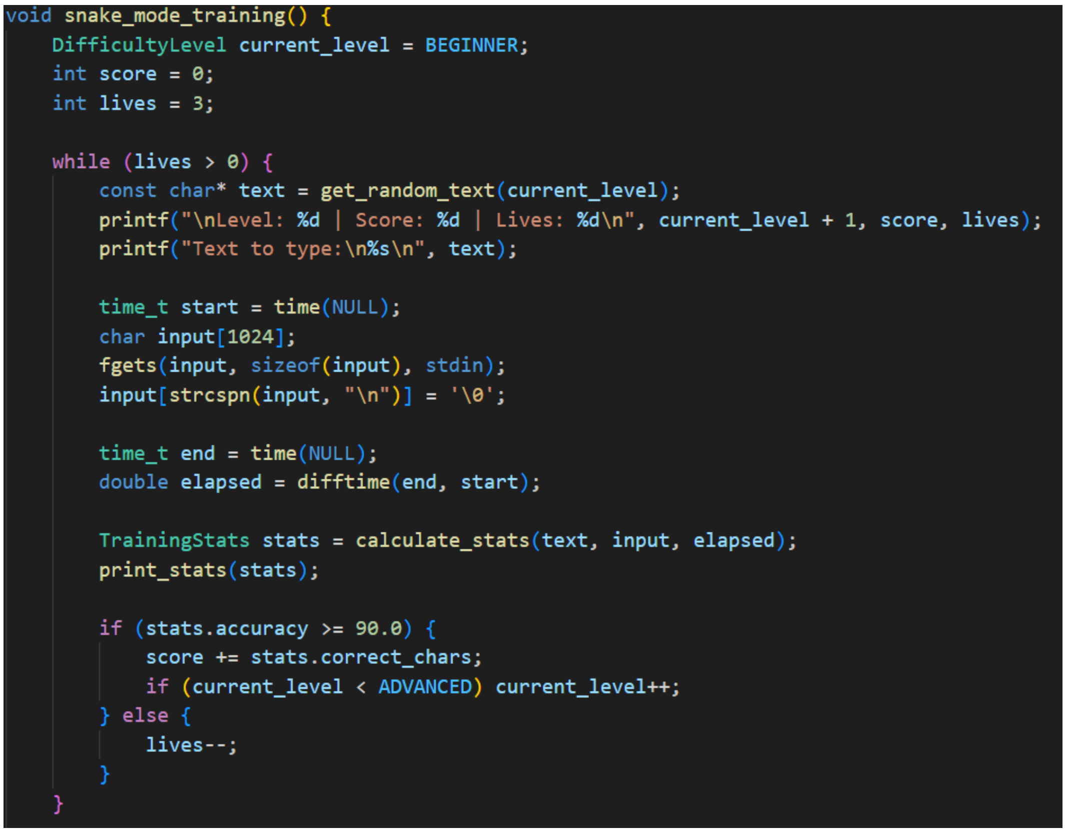
- Возможность выхода командой /q.



```c
        }

        input[strcspn(input, "\n")] = '\0';  // Удалить символ новой строки

        // Проверка на выход
        if (strcmp(input, "/q") == 0) {
            printf("Exiting Dynamic Mode early.\n");
            break;
        }
```

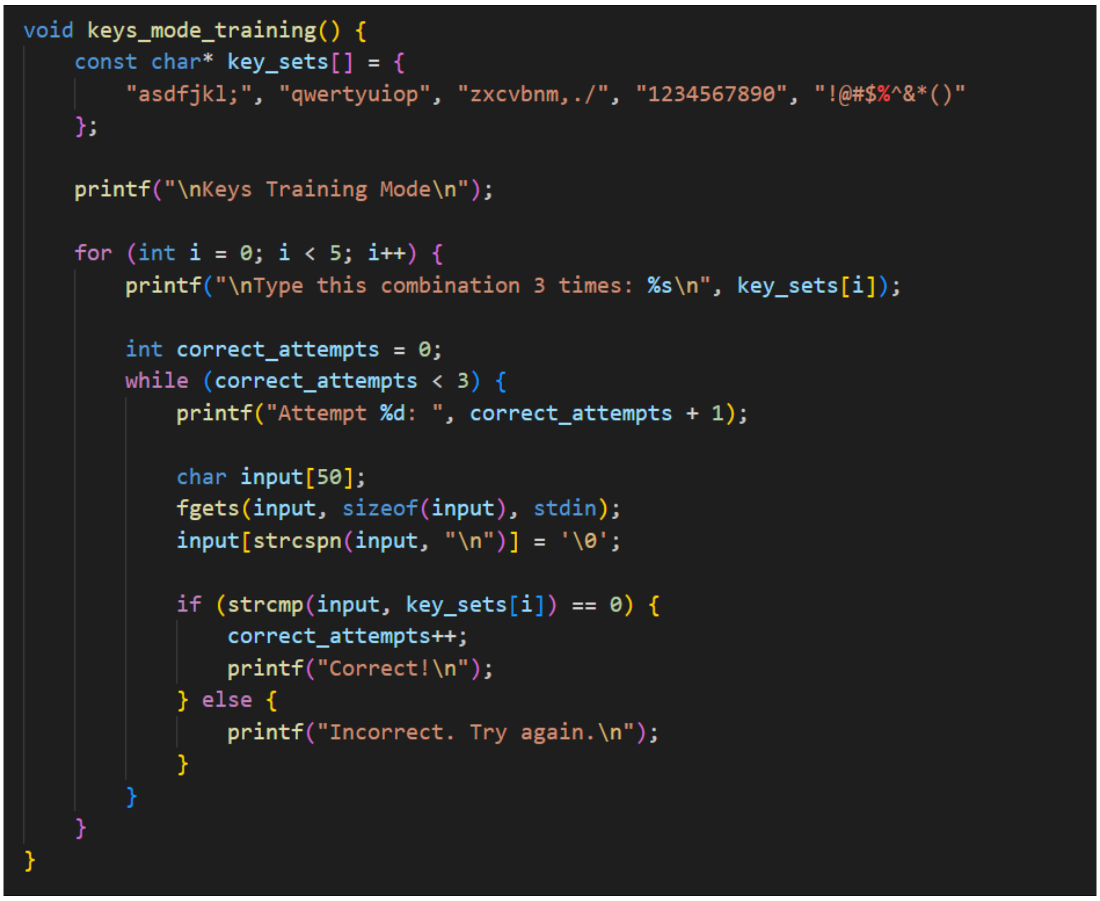
3. Режим "Змейка"
    - Прогрессивная сложность: при высокой точности уровень повышается.

- Система жизней: за ошибки снимаются жизни.
- Сохранение итогового счета.



```c
void snake_mode_training() {
    DifficultyLevel current_level = BEGINNER;
    int score = 0;
    int lives = 3;

    while (lives > 0) {
        const char* text = get_random_text(current_level);
        printf("\nLevel: %d | Score: %d | Lives: %d\n", current_level + 1, score, lives);
        printf("Text to type:\n%s\n", text);

        time_t start = time(NULL);
        char input[1024];
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0';

        time_t end = time(NULL);
        double elapsed = difftime(end, start);

        TrainingStats stats = calculate_stats(text, input, elapsed);
        print_stats(stats);

        if (stats.accuracy >= 90.0) {
            score += stats.correct_chars;
            if (current_level < ADVANCED) current_level++;
        } else {
            lives--;
        }
    }
```

4. Тренировка комбинаций клавиш
    - Пользователь тренирует набор определенных комбинаций (например, "asdfjkl;").
    - Требуется правильно ввести комбинацию 3 раза подряд.



```c
void keys_mode_training() {
    const char* key_sets[] = {
        "asdfjkl;", "qwertyuiop", "zxcvbnm,./", "1234567890", "!@#$%^&*()"
    };

    printf("\nKeys Training Mode\n");

    for (int i = 0; i < 5; i++) {
        printf("\nType this combination 3 times: %s\n", key_sets[i]);

        int correct_attempts = 0;
        while (correct_attempts < 3) {
            printf("Attempt %d: ", correct_attempts + 1);

            char input[50];
            fgets(input, sizeof(input), stdin);
            input[strcspn(input, "\n")] = '\0';

            if (strcmp(input, key_sets[i]) == 0) {
                correct_attempts++;
                printf("Correct!\n");
            } else {
                printf("Incorrect. Try again.\n");
            }
        }
    }
}
```

5. Соревновательный режим
    - Минутный марафон: набор как можно большего количества текста.
    - Сохранение результатов в таблицу рекордов.

2.2. Дополнительные функции

- Статистика
    - История тренировок с детализацией по каждой сессии.
    - Сохранение в файл `stats.txt`.
- Таблица рекордов
    - Топ-10 результатов с сортировкой по скорости.
    - Сохранение в файл `leaderboard.txt`.
- Многопоточность
    - Реализована для имитации параллельных тренировок (тестовая функция).

---

## 3. Структура проекта

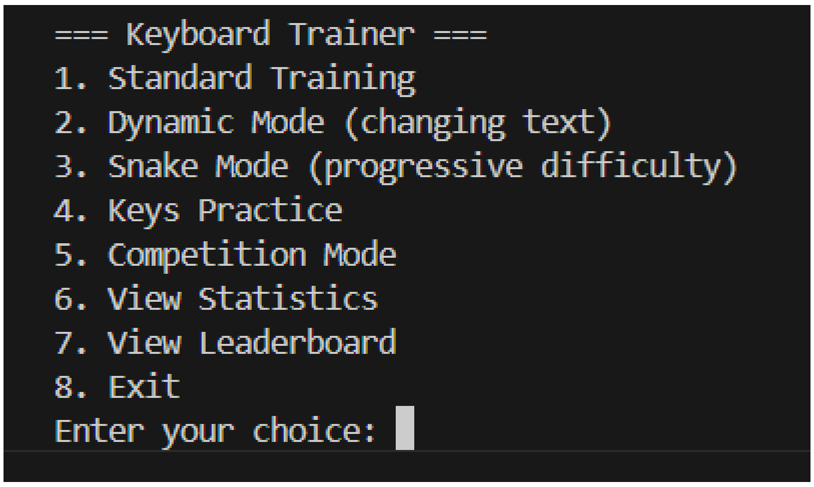
- main.c
    - Основной цикл программы, меню выбора режимов.

- keyboard_trainer.c
    - Реализация всех функций приложения.
- keyboard_trainer.h
    - Заголовочный файл с определениями структур и функций.
- test_keyboard_trainer.c
    - Юнит-тесты для проверки корректности работы функций.
- CMakeLists.txt
    - Конфигурация сборки проекта.

---

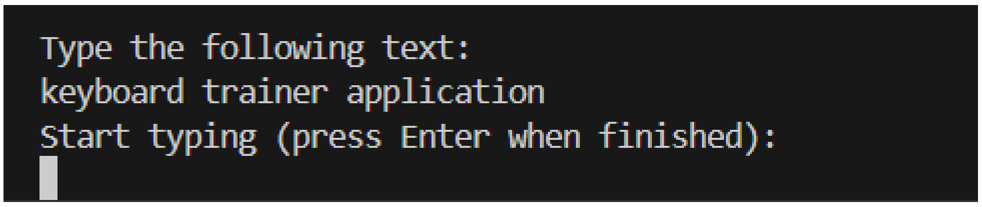
## 4. Скриншоты работы приложения

1. Главное меню



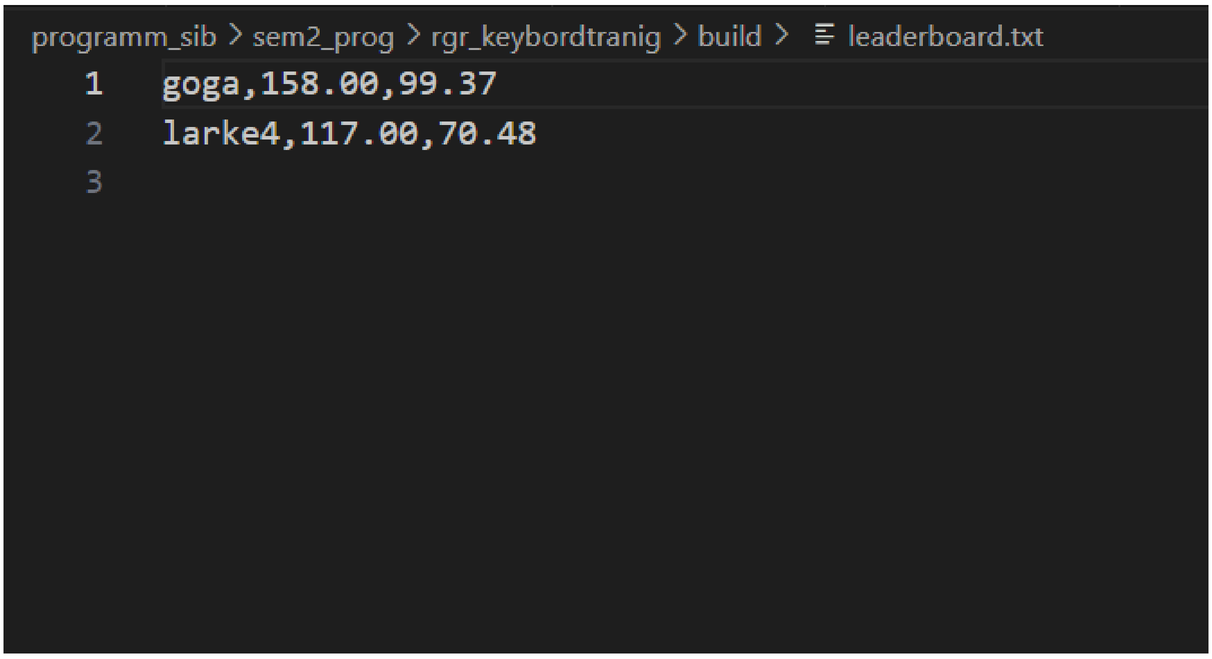
```
=== Keyboard Trainer ===
1. Standard Training
2. Dynamic Mode (changing text)
3. Snake Mode (progressive difficulty)
4. Keys Practice
5. Competition Mode
6. View Statistics
7. View Leaderboard
8. Exit
Enter your choice:
```

2. Пример тренировки в стандартном режиме



```
Type the following text:
keyboard trainer application
Start typing (press Enter when finished):
```

3. Таблица рекордов


programm_sib > sem2_prog > rgr_keybordtranig > build > leaderboard.txt

```
goga,158.00,99.37
larke4,117.00,70.48

```

---

## 5. Тестирование

Приложение было протестировано с помощью юнит-тестов:

- Проверка расчета статистики.
- Проверка генерации текста.
- Тест многопоточности.

Результаты:

```
All tests passed successfully!
```

## 6. Заключение

В ходе работы было разработано консольное приложение для тренировки печати с поддержкой нескольких режимов, статистикой и таблицей рекордов. Приложение может быть расширено добавлением новых текстов или режимов.

Список использованных технологий:

- Язык программирования C.
- Библиотеки: `pthread`, `time.h`, `string.h`.
- Система сборки CMake.

7. Листринг кода

**Keyboard_trainer.c**

```c
#include "keyboard_trainer.h"

// Тексты для тренировки
const char* beginner_texts[] = {
    "hello world programming",
    "the quick brown fox jumps",
    "learn to type faster",
    "keyboard trainer application",
    "c programming is fun",
    "practice makes perfect",
    "computer science basics",
    "algorithm and data structures",
    "github repository commit",
    "linux terminal commands"
};

const char* intermediate_texts[] = {
    "The quick brown fox jumps over the lazy dog. This sentence contains all English letters.",
    "Programming is the process of creating instructions that enable a computer to perform tasks.",
    "Memory management in C requires careful allocation and deallocation to prevent leaks.",
    "Debugging involves identifying and removing errors from computer programs.",
    "Version control systems like Git help developers collaborate on projects efficiently.",
    "Regular expressions provide a powerful way to search and manipulate text patterns.",
    "Object-oriented programming focuses on data structures containing both data and functions.",
    "The Linux kernel is written mostly in C and forms the core of operating systems.",
    "Compilers translate high-level programming languages into machine code.",
    "Multithreading allows programs to execute multiple operations concurrently."
};

const char* advanced_texts[] = {
    "In computer science, a pointer is a programming language object that stores a memory address. Pointers are used to build complex data structures and manipulate memory directly.",
    "The C programming language, developed in 1972 by Dennis Ritchie at Bell Labs, revolutionized computing by providing low-level access to memory while maintaining portability.",
    "Modern cryptography relies on mathematical algorithms to secure communications, including public-key systems like RSA and elliptic-curve cryptography.",
    "Operating system kernels handle memory management, process scheduling, device I/O, and provide the fundamental interface between hardware and software components.",
    "The TCP/IP protocol suite forms the basis of internet communications, implementing reliable packet delivery, addressing, and routing across interconnected networks.",
```

```c
    "Quantum computing leverages quantum-mechanical phenomena like superposition and entanglement to perform calculations exponentially faster than classical computers for certain problems.",
    "Distributed systems face unique challenges including network latency, partial failures, and consistency models like eventual consistency and strong consistency.",
    "Compiler optimization techniques include dead code elimination, loop unrolling, register allocation, and instruction scheduling to improve runtime performance.",
    "Functional programming paradigms emphasize immutable data and first-class functions, avoiding side effects and mutable state found in imperative programming.",
    "Computer architecture innovations like pipelining, speculative execution, and SIMD instructions have dramatically increased processor performance over decades."
};
void print_main_menu() {
    printf("\n=== Keyboard Trainer ===\n");
    printf("1. Standard Training\n");
    printf("2. Dynamic Mode (changing text)\n");
    printf("3. Snake Mode (progressive difficulty)\n");
    printf("4. Keys Practice\n");
    printf("5. Competition Mode\n");
    printf("6. View Statistics\n");
    printf("7. View Leaderboard\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
}
```

```
int get_menu_choice() {
    int choice;
    scanf("%d", &choice);
    while(getchar() != '\n');
    return choice;
}

DifficultyLevel select_difficulty() {
    printf("\nSelect difficulty:\n");
    printf("1. Beginner\n");
    printf("2. Intermediate\n");
    printf("3. Advanced\n");
    printf("Choice: ");
    return (DifficultyLevel)(get_menu_choice() - 1);
}

const char* get_random_text(DifficultyLevel level) {
    int count;
    const char** texts;

    switch(level) {
        case BEGINNER:
            texts = beginner_texts;
            count =
sizeof(beginner_texts)/sizeof(beginner_texts[0]);
```

```
            break;
        case INTERMEDIATE:
            texts = intermediate_texts;
            count =
sizeof(intermediate_texts)/sizeof(intermediate_texts[0]);
            break;
        case ADVANCED:
            texts = advanced_texts;
            count =
sizeof(advanced_texts)/sizeof(advanced_texts[0]);
            break;
        default:
            return beginner_texts[0];
    }
    return texts[rand() % count];
}

TrainingStats calculate_stats(const char* original, const
char* input, double time_elapsed) {
    TrainingStats stats = {0};
    stats.time_elapsed = time_elapsed;

    int original_len = strlen(original);
    int input_len = strlen(input);
    int min_len = original_len < input_len ? original_len :
input_len;
```

```
    for (int i = 0; i < min_len; i++) {
        if (original[i] == input[i]) {
            stats.correct_chars++;
        } else {
            stats.incorrect_chars++;
        }
    }

    if (original_len > input_len) {
        stats.incorrect_chars += original_len - input_len;
    } else if (input_len > original_len) {
        stats.incorrect_chars += input_len - original_len;
    }

    stats.speed = (stats.correct_chars / time_elapsed) * 60;
    stats.accuracy = (double)stats.correct_chars /
(stats.correct_chars + stats.incorrect_chars) * 100;

    return stats;
}

void print_stats(TrainingStats stats) {
    printf("\n--- Training Results ---\n");
    printf("Correct characters: %d\n", stats.correct_chars);
    printf("Incorrect characters: %d\n",
stats.incorrect_chars);
```

```
    printf("Time elapsed: %.2f seconds\n",
stats.time_elapsed);

    printf("Typing speed: %.2f characters per minute\n",
stats.speed);

    printf("Accuracy: %.2f%%\n", stats.accuracy);

}


void save_stats(TrainingStats stats) {

    FILE *file = fopen("stats.txt", "a");

    if (file) {

        fprintf(file, "%d,%d,%.2f,%.2f,%.2f\n",

                stats.correct_chars,

                stats.incorrect_chars,

                stats.time_elapsed,

                stats.speed,

                stats.accuracy);

        fclose(file);

    }

}


void show_statistics() {

    FILE *file = fopen("stats.txt", "r");

    if (!file) {

        printf("No statistics available yet.\n");

        return;

    }
    printf("\n--- Training History ---\n");
    printf("Correct | Incorrect | Time (s) | Speed (cpm) |
Accuracy\n");

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }

    fclose(file);
}

void* training_thread(void* args) {
    TrainingThreadArgs* targs = (TrainingThreadArgs*)args;
    const char* text = get_random_text(targs->level);

    time_t start_time = time(NULL);
    char input[1024];
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0';

    time_t end_time = time(NULL);
    double time_elapsed = difftime(end_time, start_time);

    targs->stats = calculate_stats(text, input, time_elapsed);
```

```c
    return NULL;
}

void standard_mode_training(DifficultyLevel level) {
    const char* text = get_random_text(level);
    printf("\nType the following text:\n%s\n", text);

    time_t start_time = time(NULL);
    printf("Start typing (press Enter when finished):\n");

    char input[1024];
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0';

    time_t end_time = time(NULL);
    double time_elapsed = difftime(end_time, start_time);

    TrainingStats stats = calculate_stats(text, input,
time_elapsed);
    print_stats(stats);
    save_stats(stats);
}

void dynamic_mode_training(DifficultyLevel level) {
    printf("\nDynamic Mode: Text will appear immediately after
each input\n");
```

```
    printf("Type '/q' at any time to quit.\n");


    int total_correct = 0, total_incorrect = 0;

    double total_time = 0;

    int rounds = 0;


    // Время на раунд в зависимости от уровня

    int round_seconds;

    switch(level) {

        case BEGINNER:      round_seconds = 30; break;

        case INTERMEDIATE:  round_seconds = 25; break;

        case ADVANCED:      round_seconds = 20; break;

        default:            round_seconds = 30; break;

    }


    while (1) {

        const char* text = get_random_text(level);

        printf("\nNew text (%d seconds to complete):",
round_seconds);

        printf("\n%s\n", text);


        time_t round_start = time(NULL);

        char input[1024];

        if (!fgets(input, sizeof(input), stdin)) {

            printf("Error reading input.\n");

            break;
        }


        input[strcspn(input, "\n")] = '\0';  // Удалить символ
новой строки


        // Проверка на выход

        if (strcmp(input, "/q") == 0) {

            printf("Exiting Dynamic Mode early.\n");

            break;

        }


        time_t round_end = time(NULL);

        double round_time = difftime(round_end, round_start);


        // Если время больше лимита, засчитываем как ошибку

        if (round_time > round_seconds) {

            printf("Time limit exceeded!\n");

        }


        TrainingStats stats = calculate_stats(text, input,
round_time);

        print_stats(stats);


        total_correct += stats.correct_chars;

        total_incorrect += stats.incorrect_chars;

        total_time += round_time;
        rounds++;
    }

    if (rounds > 0) {
        TrainingStats final_stats = {
            total_correct,
            total_incorrect,
            total_time,
            (total_correct / total_time) * 60,
            (double)total_correct / (total_correct +
total_incorrect) * 100
        };
        printf("\n--- Final Dynamic Mode Results ---\n");
        print_stats(final_stats);
        save_stats(final_stats);
    } else {
        printf("No rounds completed.\n");
    }
}

void snake_mode_training() {
    DifficultyLevel current_level = BEGINNER;
    int score = 0;
    int lives = 3;

    while (lives > 0) {
        const char* text = get_random_text(current_level);
        printf("\nLevel: %d | Score: %d | Lives: %d\n",
current_level + 1, score, lives);
        printf("Text to type:\n%s\n", text);

        time_t start = time(NULL);
        char input[1024];
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0';

        time_t end = time(NULL);
        double elapsed = difftime(end, start);

        TrainingStats stats = calculate_stats(text, input,
elapsed);
        print_stats(stats);

        if (stats.accuracy >= 90.0) {
            score += stats.correct_chars;
            if (current_level < ADVANCED) current_level++;
        } else {
            lives--;
        }
    }

    printf("\nGame over! Final score: %d\n", score);
    save_snake_score(score);
}

void save_snake_score(int score) {
    FILE* file = fopen("snake_scores.txt", "a");
    if (file) {
        fprintf(file, "%d\n", score);
        fclose(file);
    }
}

void keys_mode_training() {
    const char* key_sets[] = {
        "asdfjkl;", "qwertyuiop", "zxcvbnm,./", "1234567890",
"!@#$%^&*()"
    };

    printf("\nKeys Training Mode\n");

    for (int i = 0; i < 5; i++) {
        printf("\nType this combination 3 times: %s\n",
key_sets[i]);

        int correct_attempts = 0;
        while (correct_attempts < 3) {
            printf("Attempt %d: ", correct_attempts + 1);
            char input[50];
            fgets(input, sizeof(input), stdin);
            input[strcspn(input, "\n")] = '\0';

            if (strcmp(input, key_sets[i]) == 0) {
                correct_attempts++;
                printf("Correct!\n");
            } else {
                printf("Incorrect. Try again.\n");
            }
        }
    }
}

void competition_mode() {
    printf("\nCompetition Mode\n");
    printf("Enter your name: ");

    char name[50];
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = '\0';

    printf("\nYou have 1 minute to type as much as
possible!\n");
    printf("Press Enter to start...");
```

```c
getchar();


time_t start = time(NULL);

int total_chars = 0;

int correct_chars = 0;

const char* text = get_random_text(INTERMEDIATE);


printf("\nText to type:\n%s\n", text);


while (difftime(time(NULL), start) < 60) {

    char input[1024];

    fgets(input, sizeof(input), stdin);

    input[strcspn(input, "\n")] = '\0';


    int len = strlen(input);

    total_chars += len;


    for (int i = 0; i < len && i < strlen(text); i++) {

        if (input[i] == text[i]) correct_chars++;

    }


    text = get_random_text(INTERMEDIATE);

    printf("\nNew text:\n%s\n", text);

}
```

```
    double accuracy = (double)correct_chars / total_chars *
100;

    double speed = correct_chars;


    printf("\nTime's up!\nCorrect characters: %d\nAccuracy:
%.2f%%\n", correct_chars, accuracy);

    save_competition_result(name, speed, accuracy);

    show_leaderboard();

}


void save_competition_result(const char* name, double speed,
double accuracy) {

    FILE* file = fopen("leaderboard.txt", "a");

    if (file) {

        fprintf(file, "%s,%.2f,%.2f\n", name, speed,
accuracy);

        fclose(file);

    }

}


void show_leaderboard() {

    FILE* file = fopen("leaderboard.txt", "r");

    if (!file) {

        printf("No leaderboard records yet.\n");

        return;

    }
    UserRecord records[100];

    int count = 0;

    char line[256];


    while (fgets(line, sizeof(line), file) && count < 100) {

        char* token = strtok(line, ",");

        strcpy(records[count].name, token);

        records[count].best_speed = atof(strtok(NULL, ","));

        records[count].best_accuracy = atof(strtok(NULL,
"\n"));

        count++;

    }

    fclose(file);


    // Сортировка по скорости

    for (int i = 0; i < count - 1; i++) {

        for (int j = 0; j < count - i - 1; j++) {

            if (records[j].best_speed <
records[j+1].best_speed) {

                UserRecord temp = records[j];

                records[j] = records[j+1];

                records[j+1] = temp;

            }

        }

    }
    printf("\n--- LEADERBOARD (Top 10) ---\n");
    printf("Rank | Name              | Speed (cpm) |
Accuracy\n");
    for (int i = 0; i < (count < 10 ? count : 10); i++) {
        printf("%2d   | %-14s | %9.2f   | %.2f%%\n",
               i+1, records[i].name, records[i].best_speed,
records[i].best_accuracy);
    }
}


void start_selected_mode(int choice) {
    DifficultyLevel level;
    switch (choice) {
        case 1: standard_mode_training(select_difficulty());
break;
        case 2: dynamic_mode_training(select_difficulty());
break;
        case 3: snake_mode_training(); break;
        case 4: keys_mode_training(); break;
        case 5: competition_mode(); break;
        case 6: show_statistics(); break;
        case 7: show_leaderboard(); break;
    }
}
```

Keyboard_trainer.h

```
#ifndef KEYBOARD_TRAINER_H
```

```c
#define KEYBOARD_TRAINER_H


#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <ctype.h>

#include <pthread.h>

#include <unistd.h>

#include <math.h>


typedef enum {

    BEGINNER,

    INTERMEDIATE,

    ADVANCED

} DifficultyLevel;


typedef enum {

    STANDARD_MODE,

    DYNAMIC_MODE,

    SNAKE_MODE,

    KEYS_MODE,

    COMPETITION_MODE

} TrainingMode;
```

```
typedef struct {
    int correct_chars;
    int incorrect_chars;
    double time_elapsed;
    double speed;
    double accuracy;
} TrainingStats;

typedef struct {
    char name[50];
    double best_speed;
    double best_accuracy;
} UserRecord;

typedef struct {
    DifficultyLevel level;
    TrainingStats stats;
} TrainingThreadArgs;

// Основные функции
void print_main_menu();
int get_menu_choice();
DifficultyLevel select_difficulty();
void start_selected_mode(int choice);
```

```
// Режимы тренировки
void standard_mode_training(DifficultyLevel level);
void dynamic_mode_training(DifficultyLevel level);
void snake_mode_training();
void keys_mode_training();
void competition_mode();

// Вспомогательные функции
const char* get_random_text(DifficultyLevel level);
TrainingStats calculate_stats(const char* original, const
char* input, double time_elapsed);
void print_stats(TrainingStats stats);
void save_stats(TrainingStats stats);
void show_statistics();
void save_snake_score(int score);
void save_competition_result(const char* name, double speed,
double accuracy);
void show_leaderboard();

// Многопоточность
void* training_thread(void* args);

#endif
```

tests_keyboard_trainer.c

```
#include "keyboard_trainer.h"
```

```c
#include <assert.h>

#include <string.h>

#include <math.h>


// Вспомогательная функция для сравнения с погрешностью

int double_eq(double a, double b) {

    return fabs(a - b) < 1e-6;

}


// --- Тест стандартного режима ---

void test_standard_mode() {

    const char* original = "hello world";

    const char* input_correct = "hello world";

    const char* input_with_errors = "hallo warld";


    TrainingStats stats1 = calculate_stats(original,
input_correct, 10.0);

    assert(stats1.correct_chars == 11);

    assert(stats1.incorrect_chars == 0);

    assert(double_eq(stats1.accuracy, 100.0));


    TrainingStats stats2 = calculate_stats(original,
input_with_errors, 10.0);

    assert(stats2.correct_chars == 9);  //
'h','e','l','l','o',' ','w','o','r' совпадают

    assert(stats2.incorrect_chars == 2); // 'l' vs 'a', 'l' vs
'd'
```

```
    assert(fabs(stats2.accuracy - 81.818182) < 0.01); // ~9/11
= 81.82%
}


// --- Ускоренный тест динамического режима ---
void test_fast_dynamic_mode() {
    DifficultyLevel level = BEGINNER;
    int total_correct = 0, total_incorrect = 0;
    double total_time = 0;

    for (int i = 0; i < 3; i++) {
        const char* text = get_random_text(level);
        char input[1024];
        strcpy(input, text); // идеальный ввод

        TrainingStats stats = calculate_stats(text, input,
5.0);
        total_correct += stats.correct_chars;
        total_incorrect += stats.incorrect_chars;
        total_time += 5.0;
    }

    assert(total_incorrect == 0);
    assert(double_eq(total_time, 15.0));
}
```

```
// --- Тест змейки ---
void test_snake_mode() {
    DifficultyLevel level = BEGINNER;
    int score = 0;
    int lives = 3;

    TrainingStats good = {50, 2, 60, 50.0, 96.15};
    if (good.accuracy >= 90.0) {
        score += good.correct_chars;
        level++;
    }
    assert(score == 50);
    assert(level == INTERMEDIATE);

    TrainingStats bad = {30, 10, 60, 30.0, 75.0};
    if (bad.accuracy < 90.0) {
        lives--;
    }
    assert(lives == 2);
}

// --- Тест клавишных комбинаций ---
void test_keys_mode() {
    const char* combo = "asdfjkl;";
    char input[32];
    strcpy(input, combo);
    assert(strcmp(input, combo) == 0);

    strcpy(input, "asdfjkl,");
    assert(strcmp(input, combo) != 0);
}

// --- Тест таблицы рекордов ---
void test_leaderboard() {
    UserRecord records[3] = {
        {"Alice", 150.5, 98.2},
        {"Bob", 120.0, 95.5},
        {"Charlie", 180.0, 97.8}
    };

    // Сортировка по скорости
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2 - i; j++) {
            if (records[j].best_speed <
records[j+1].best_speed) {
                UserRecord temp = records[j];
                records[j] = records[j+1];
                records[j+1] = temp;
            }
        }
    }

    assert(strcmp(records[0].name, "Charlie") == 0);
    assert(strcmp(records[1].name, "Alice") == 0);
    assert(strcmp(records[2].name, "Bob") == 0);
}

// --- Тест многопоточности ---
void* mock_training(void* args) {
    TrainingThreadArgs* targs = (TrainingThreadArgs*)args;
    const char* text = get_random_text(targs->level);
    char input[1024];
    strcpy(input, text);

    time_t start = time(NULL);
    sleep(1); // имитация задержки
    time_t end = time(NULL);

    double elapsed = difftime(end, start);
    targs->stats = calculate_stats(text, input, elapsed);
    return NULL;
}

void test_threading() {
    TrainingThreadArgs args = {BEGINNER};
```

```c
    pthread_t thread;

    pthread_create(&thread, NULL, mock_training, &args);
    pthread_join(thread, NULL);

    assert(args.stats.correct_chars > 0);
    assert(args.stats.time_elapsed >= 1.0);
}

// --- Основной блок запуска тестов ---
int main() {
    printf("Running fast tests...\n");

    test_standard_mode();
    test_fast_dynamic_mode();
    test_snake_mode();
    test_keys_mode();
    test_leaderboard();
    test_threading();

    printf("All tests passed successfully!\n");
    return 0;
}
```

main.c

```c
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <time.h>
#include "keyboard_trainer.h"

int main() {
    srand(time(NULL));

    int choice;
    do {
        print_main_menu();
        choice = get_menu_choice();

        if (choice >= 1 && choice <= 7) {
            start_selected_mode(choice);
        } else if (choice != 8) {
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 8);

    printf("Exiting...\n");
    return 0;
}
```