

**Міністерство освіти і науки України**  
**Національний технічний університет України**  
**«Київський політехнічний інститут ім. Ігоря Сікорського»**  
**Факультет інформатики та обчислювальної техніки**  
**Кафедра обчислювальної техніки**

**КУРСОВА РОБОТА**  
**з дисципліни «Методології та технології розробки ПЗ» на тему**  
**«Ігровий застосунок у жанрі шутер»**

**ВИКОНАВ: Студент 2 курсу ФІОТ групи ІП-04, № у списку (варіант) - 11**

**Кравченко Владислав**

**ПЕРЕВІРИВ: ас. Ковальчук О. М.**

**Київ - 2022**

### **Короткий зміст:**

*Консольний застосунок для тренування швидкого друку (для безінтернетних часів), орієнтований на сімейство операційних систем Лінукс (проект у розробці).*

### **Використані інструменти/бібліотеки:**

*Мова програмування Сі (C programming language).*

*Бібліотека консольної графіки Неокюрсис (ncurses cli library).*

### **Особливості проекту:**

*Чистота, простота, та мінімалізм.*

### **Висновок:**

*Працювати з мовою програмування Сі хоч і важко, але приємно.*

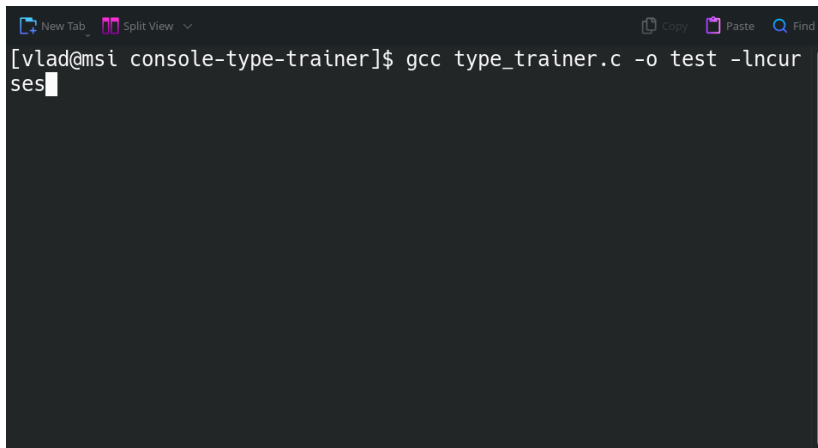
*Результатом є дуже “легка” (в плані пам’яті) та інтуїтивна програма.*

*Бібліотека Неокюрсис доволі обширна і має значний потенціал. Втім, було б бажано, щоб вона була краще задокументована (доволі важко розібратися з усім самотужки). Інтерфейс програмної взаємодії (API) консолі Лінуксу натомість жахливий, працювати з ним суцільне пекло (на щастя Неокюрсис значною мірою полегшує діль, але не може погасити його повною мірою). Загалом, в результаті роботи над цим проектом я багато чому навчився та багато чого усвідомив. Втім, я змушений був його призупинити в зв’язку з нестачею часу. Але сподіваюсь уже скоро я зможу повернутись до нього знову.*

### **Посилання на гітхаб проекту:**

<https://github.com/cyberlord-coder-228/console-type-trainer>

### ***Скриншоти виконання програми:***




```
[vlad@msi console-type-trainer]$ gcc type_trainer.c -o test -lncurses
```

***(Процес компіляції дуже простий (для будь-кого знайомого з gcc) та не вимагає жодних залежностей)***



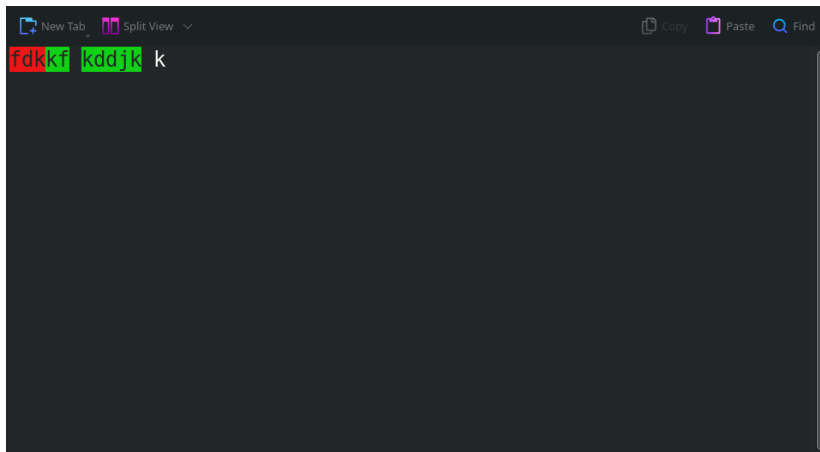
```
[vlad@msi console-type-trainer]$ ./test
```

***(Запуск програми через консоль надзвичайно простий та інтуїтивний)***

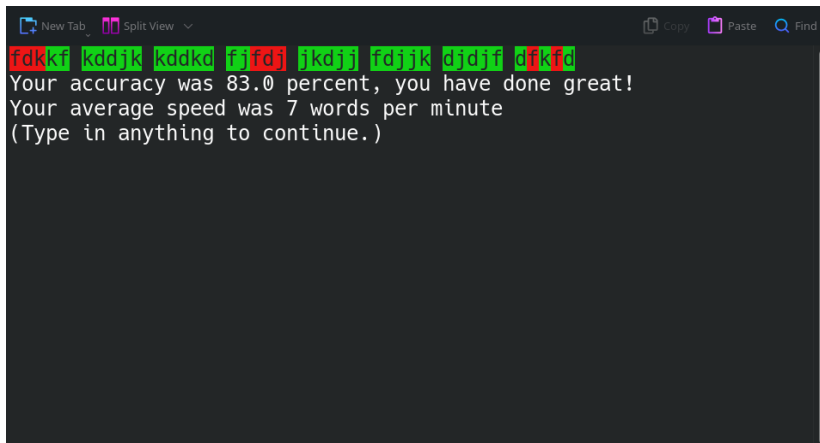


```
Start basic exercise
Select level
Exit
```

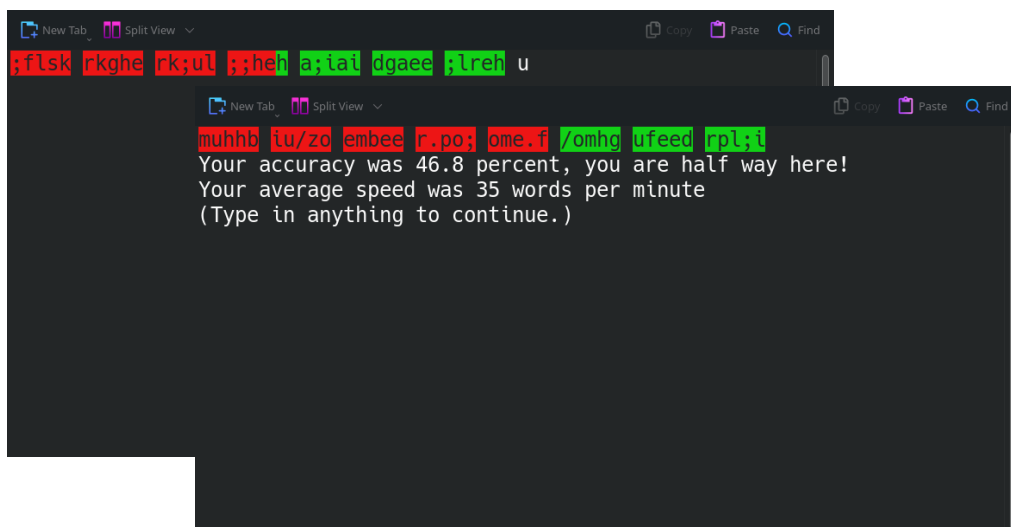
***(Початкове меню (на даний момент все ще у розробці))***



*(Початок виконання програми [зелені символи - правильно введені, червоні - помилкові], всі символи генеруються випадковим чином залежно від складності)*



*(Повідомлення після проходження “уроку”)*



*(Складність поступово збільшується додаванням нових символів)*

### ***Код програми:***

type\_trainer.c (основна і самодостатня частина)

```
#include <ncurses.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>

#define WINDOW_WIDTH 63
#define WINDOW_HEIGHT 27

const int WPM_EQUATION_CONSTANT = 12;

enum
{
    RED_BACKGROUND_NUM    = 1,
    GREEN_BACKGROUND_NUM   = 2
};

char* LATIN_LETTERS = "abcdefghijklmnopqrstuvwxyz";
    // 26 symbols
char* LEARNER_SEQUENCE = "fjdksla;ghrueiwoqptyvmc,x.z/bn"; //
30 symbols

struct TimeHolder
{
    struct timeval start;
    struct timeval end;
};

char get_rand_letter(char* arr, int arr_length)
{
    return arr[rand() % arr_length];
}
```

```

float gibberish_words_exercise(
    WINDOW* active_window,
    int amount_of_symbols,
    int word_length,
    char* letters_to_practise,
    int letters_arr_length,
    int start_line
)
{
    int mistakes = 0;

    for (
        int i = 0;
        i < amount_of_symbols;
        i += (++i % (word_length + 1) == word_length)
    )
    {
        char rand_char = get_rand_letter(
            letters_to_practise,
            letters_arr_length
        );
        mvwaddch(active_window, start_line, i, rand_char);
        wrefresh(active_window);

        char user_input = getch();

        int color_num;
        if (rand_char == user_input)
        {
            color_num = GREEN_BACKGROUND_NUM;
        }
        else
        {
            mistakes++;
            color_num = RED_BACKGROUND_NUM;
        }

        wattrset(active_window, COLOR_PAIR(color_num));
    }
}

```

```

        mvwaddch(active_window, start_line, i, rand_char);
        wattroff(active_window, COLOR_PAIR(color_num));
        wrefresh(active_window);
    }

    return (float)(amount_of_symbols - mistakes) /
(float)amount_of_symbols;
}

void print_how_good_user_was(
    WINDOW* active_window,
    float correct_ratio,
    int wpm
)
{
    char* postfix;
    if (correct_ratio == 1.0)          postfix = ".
Perfect!";
    else if (correct_ratio > 0.8) postfix = "you have done
great!";
    else if (correct_ratio > 0.6) postfix = "you have done
quite good!";
    else if (correct_ratio > 0.4) postfix = "you are half way
here!";
    else if (correct_ratio > 0.2) postfix = "keep
practicing!";
    else if (correct_ratio > 0.0) postfix = "you can do
better!";
    else if (correct_ratio == .0) postfix = "did you even
try?";
    else postfix = ". Wait what? How that`s even possible?";

    wprintw(
        active_window,
        "\nYour accuracy was %.1f percent, %s",
        correct_ratio*100,
        postfix
    );
};

```

```

        wprintw(
            active_window,
            "\nYour average speed was %d words per minute",
            wpm
        );
        wrefresh(active_window);
    }

void offer_next_screen(WINDOW* active_window)
{
    wprintw(active_window, "\n(Type in anything to
continue.)");
    wrefresh(active_window);
    getch();
    clear();
}

void test(
    WINDOW* active_window,
    int symbols_in_exercise,
    int word_length
)
{
    for (int i = 1; i < 16; i++)
    {
        struct TimeHolder timer;
        gettimeofday(&timer.start, NULL);

        float ratio = gibberish_words_exercise(
            active_window,
            symbols_in_exercise,
            word_length,
            LEARNER_SEQUENCE,
            i * 2,    // amount of practice letters growths
with exery iteration
                    0    // always print exercise on a 0-th
line
        );
    }
}

```



```

        gettimeofday(&timer.end, NULL);

        int time_spent_seconds = timer.end.tv_sec -
timer.start.tv_sec;

        int user_speed_wpm = (int)(
            WPM_EQUATION_CONSTANT
            * (float)symbols_in_exercise
            / (float)time_spent_seconds
        );

        print_how_good_user_was(active_window, ratio,
user_speed_wpm);
        offer_next_screen(active_window);
    }
}

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

    initscr(); // Starts ncurses

    WINDOW* in_game_window = newwin(
        WINDOW_HEIGHT,
        WINDOW_WIDTH,
        0,
        0
    );

    cbreak(); // Makes typed stuff
immediately available
    noecho(); // Keyboard input not
printed
    scrollok(stdscr, FALSE); // Scroll disabled
    keypad(in_game_window, TRUE); // Getch returns special
stuff from arrows

```

```

    curs_set(0);

    start_color();                // Enables colors
    init_pair(RED_BACKGROUND_NUM, COLOR_BLACK, COLOR_RED);
    init_pair(GREEN_BACKGROUND_NUM, COLOR_BLACK,
COLOR_GREEN);

    const int symbols_in_test_exercise = 47;
    const int test_word_length = 5;
    test(in_game_window, symbols_in_test_exercise,
test_word_length);

    delwin(in_game_window);
    endwin();                    // Ends ncurses mode

    return 0;
}

```

menu.c (*сиря частина*)

```

#include<ncurses.h>

#define AMOUNT_OF_MENU_OPTIONS 3
#define LONGEST_OPTION_LENGTH 20

#define TOP_PADDING 0
#define LEFT_PADDING 0

#define WINDOW_WIDTH 63
#define WINDOW_HEIGHT 27

#define MY_KEY_ENTER 10

const char LIST[AMOUNT_OF_MENU_OPTIONS][LONGEST_OPTION_LENGTH]
= {
    "Start basic exercise",
    "Select level",
    "Exit"
}

```

```

};

enum
{
    BASIC_MENU_OPTION = 0,
    LEVEL_SELECT_MENU_OPTION = 1,
    EXIT_MENU_OPTION = 2
};

void print_menu(WINDOW* active_window)
{
    wattron(active_window, A_STANDOUT);
    for (int i = 0; i < AMOUNT_OF_MENU_OPTIONS; i++)
    {
        mvwprintw(active_window, i, LEFT_PADDING, "%.20s",
LIST[i]);
        wattroff(active_window, A_STANDOUT);
    }
    wrefresh(active_window);
}

void loop_menu(WINDOW* active_window, int sel_line)
{
    int selected_line = sel_line;
    while(1)
    {
        int user_input = wgetch(active_window);

        // deselect previous selection
        wattroff(active_window, A_STANDOUT);
        mvwprintw(
            active_window,
            selected_line,
            LEFT_PADDING,
            "%s",
            LIST[selected_line]
        );
        wrefresh(active_window);
    }
}

```

```

switch(user_input)
{
    case MY_KEY_ENTER:
    {
        switch (selected_line)
        {
            case BASIC_MENU_OPTION:
            {
                break;
            }
            case LEVEL_SELECT_MENU_OPTION:
            {
                break;
            }
            case EXIT_MENU_OPTION:
            {
                delwin(active_window);
                endwin();
                return;
            }
        }
    }
    case KEY_UP:
    {
        selected_line = (--selected_line >= 0)
            ? selected_line :
AMOUNT_OF_MENU_OPTIONS - 1;
        break;
    }
    case KEY_DOWN:
    {
        selected_line = (++selected_line <
AMOUNT_OF_MENU_OPTIONS)
            ? selected_line : 0;
        break;
    }
}

```

```

        // highlight selected item
        watttrn(active_window, A_STANDOUT);
        mvwprintw(
            active_window,
            selected_line,
            LEFT_PADDING,
            "%s",
            LIST[selected_line]
        );
        wrefresh(active_window);
    }
}

int main()
{
    initscr();                                // starts ncurses

    WINDOW* menu_window = newwin(
        WINDOW_HEIGHT,
        WINDOW_WIDTH,
        LEFT_PADDING,
        TOP_PADDING
    );

    noecho();                                // keyboard input not
printed
    keypad(menu_window, TRUE);                // getch returns
special stuff from arrows
    curs_set(0);                              // hide cursor

    print_menu(menu_window);
    loop_menu(menu_window, 0);

    delwin(menu_window);
    endwin();
}

```

```
    return 0;  
}
```