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 «Игра “Ним”»

Выполнил:
ст. гр. ИП-313
Широков К. В.

Проверил:
ст. преподаватель Токмашева Е. И.

Новосибирск, 2024

Содержание

Введение и постановка задачи

Цель курсовой работы

Целью курсовой работы является разработка и реализация программы для игры "Ним" с использованием современных инструментов разработки программного обеспечения. В процессе выполнения работы должны быть изучены и применены на практике:

1) Принцип работы и стратегии игры «Ним».

2) Создание и настройка Makefile для автоматизации сборки проекта.

3) Настройка конфигурационных файлов `.gitignore` и `.clang-format` для упрощения разработки и поддержания стандарта кода.

4) Настройка CI для автоматического тестирования и сборки проекта при каждом изменении в коде.

Задание на курсовую работу

- Изучить правила и стратегии игры «Ним».
- Оформить репозиторий (README.md, описание проекта, инструкции по запуску).
- Создать и настроить файл `.gitignore` для исключения из репозитория ненужных файлов и директорий.
- Создать и настроить файл `.clang-format` для автоматического форматирования кода согласно выбранному стилю.
- Написать Makefile для автоматизации сборки и тестирования проекта.
- Обеспечить выполнение всех необходимых команд для сборки проекта и запуска тестов с использованием Makefile.
- Настроить файл конфигурации `.gitlab-ci.yml`, чтобы автоматизировать процесс сборки и тестирования проекта при каждом коммите в репозиторий.
- Реализовать консольную версию игры «Ним» на языке C.
- Предусмотреть уровни сложности игры, выбор количества спичек в кучках, выбор первого хода.
- Покрыть приложение тестами.
- Подготовить отчет по выполненной работе, включающий описание игры, стратегии, алгоритмы, структуру программы, конфигурационные файлы, Makefile и настройку CI.

Техническое задание

Словарь терминов. Описание основных объектов и сокращений

Игра "Ним" — логическая игра, в которой два игрока поочередно берут предметы из нескольких куч. Побеждает тот, кто возьмет последний предмет.

Куча — группа предметов, из которой игрок может взять определенное количество предметов.

Ход — действие игрока, заключающееся в выборе количества предметов из кучи.

Победа — состояние игры, при котором один из игроков берет последний предмет из последней кучи.

Спички — предметы, которые игрок будет доставать из кучи.

Назначение разработки

Целью разработки является создание компьютерной версии игры "Ним" на языке программирования С. Игра должна обеспечивать возможность играть против компьютера.

Технические положения

- Язык программирования: С
- Совместимость: Приложение должно работать на всех операционных системах.
- Графический интерфейс: Игра будет реализована в консоли.
- Ввод/вывод: Пользовательский ввод будет осуществляться через консоль, возможен будет выбор кучки, количество элементов в каждой кучке, уровень сложности игры, вариант игры и первый ход.
- Исходный код: Должен быть хорошо комментирован и организован, соблюдая принципы чистого кода.

Логика работы

1) Игра начинается с описания правил игры.
2) Далее игрок выбирает уровень сложности и количество спичек в каждой кучке (рандомное или статическое).
3) Затем можно выбрать, кто первый ходит.
4) Игроки (человек и компьютер) поочередно делают ходы, выбирая номер кучки и количество спичек для взятия.
5) Игра продолжается до тех пор, пока не будет достигнуто состояние победы.
6) После завершения игры выводится информация о победителе.
7) Возможность продолжить играть дальше или выйти из игры.

Требования к функциональным характеристикам

Программа должна обеспечивать возможность выполнения нижеперечисленных функций:

1. Модель игры:
   - Структура данных для представления текущего состояния игры, включая количество кучек и количество спичек в каждой кучке.
   - Логика для выполнения ходов игроков и проверки условий завершения игры.
2. Интерфейс ввода-вывода:
   - Функции для отображения текстовой информации о текущем состоянии игры (количество спичек в кучках, чей ход и т.д.).
   - Возможность ввода данных от пользователя (выбор кучек для хода).
3. Логика игры:
   - Функции для реализации правил игры "Ним", включая определение победителя и обработку некорректных ходов.
4. Управление потоком игры:
   - Функции для управления ходом игры (передача хода между игроками, определение конца игры и т.д.).
5. Обработка ошибок:
   - Механизмы для обнаружения и обработки ошибок, таких как некорректный ввод пользователя или непредвиденные ситуации в ходе игры.

Интерфейс приложения

Режим работы: Интерактивный, где пользователь взаимодействует с приложением через консоль.

Элементы интерфейса:

- Краткое описание правил игры. Перед началом игры пользователь может ознакомиться с правилами.
- Поле игры, где отображаются кучки со спичками.
- Панель с информацией о текущем состоянии игры (чей ход, количество оставшихся элементов и т.д.).
- Команды для совершения хода и перехода к следующему шагу игры. Аргументы командной строки: В данной реализации игры "Ним" аргументы командной строки могут быть использованы для запуска игры с определенными параметрами, такими как количество кучек и начальное количество элементов в каждой кучке.

Панель администрирования / Настройки

Поскольку игра "Ним" не требует настроек или администрирования, панель администрирования и настройки отсутствуют.

Описание выполненного проекта

Разработанный проект представляет собой готовую компьютерную версию игры "Ним", реализованную на языке программирования C. Игра позволяет играть против компьютера, выбрав уровень сложности и вариацию игры. Проект включает в себя модульные тесты, созданные с использованием библиотеки CTest, для проверки основных функций игры и валидации ввода данных.

При разработке проекта были созданы следующие файлы:

1. `.gitignore`: Файл, определяющий игнорируемые файлы и папки для системы контроля версий Git. В нем указаны файлы, которые не должны быть включены в репозиторий, например, временные файлы, объектные файлы компиляции и исполняемые файлы.
2. `.clang-format`: Конфигурационный файл, определяющий стиль форматирования кода с помощью инструмента clang-format. Он обеспечивает единообразие кода в проекте, определяя правила отступов, расстановки фигурных скобок и прочих аспектов форматирования.
3. `.gitlab-ci.yml`: Файл конфигурации для настройки процесса непрерывной интеграции (CI) с использованием GitLab CI/CD. Здесь определены шаги сборки, тестирования и развертывания проекта, которые выполняются автоматически при каждом коммите или пуше в репозиторий.
4. `Makefile`: Файл с инструкциями для сборки проекта с помощью утилиты make. Он определяет зависимости между исходными файлами, объектными файлами и исполняемыми файлами, а также содержит правила для компиляции и сборки проекта.

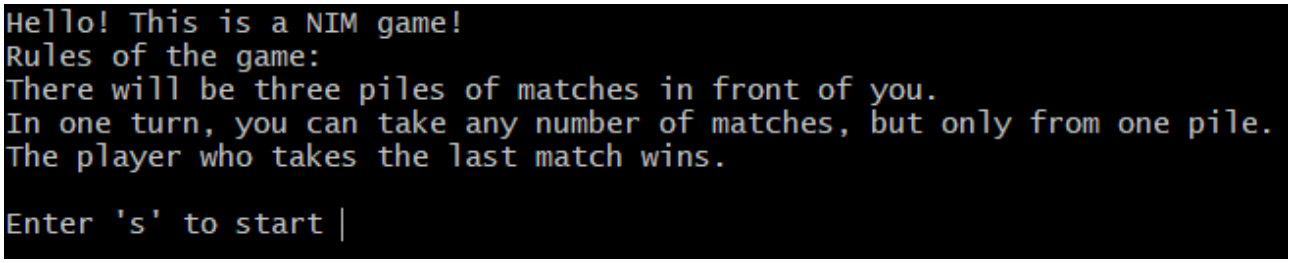
Также в проекте присутствует файл README.md, содержащий инструкции по сборке и запуску приложения.

В проекте были реализованы следующие функции:

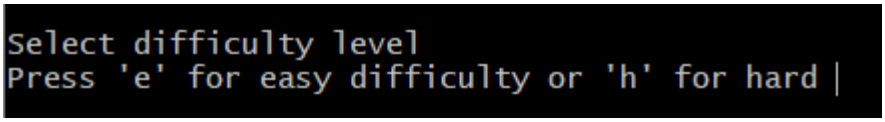
1. `initializeStaticBoard`: Функция инициализации игрового поля с фиксированным количеством матчей в каждой строке.
2. `initializeRandomBoard`: Функция инициализации игрового поля с случайным количеством матчей в каждой строке.
3. `printBoard`: Функция вывода текущего состояния игрового поля на экран.
4. `removeMatches`: Функция удаления указанного количества матчей из указанной строки игрового поля.
5. `gameOver`: Функция проверки окончания игры путем определения отсутствия матчей на игровом поле.
6. `computerTurnWin`: Функция оптимального хода компьютера, который основывается на вычислении Nim-суммы и выборе такого хода, чтобы оставить сопернику невыигрышную позицию.
7. `computerTurnRandom`: Функция случайного хода компьютера, когда он выбирает случайную строку и случайное количество матчей для удаления.
8. `playGame`: Функция, осуществляющая ходы игроков в игре и определяющая победителя.
9. `main`: Основная функция программы, которая управляет ходом игры, взаимодействием с пользователем и вызывает другие функции в зависимости от выбора пользователя.

Демонстрация работы:

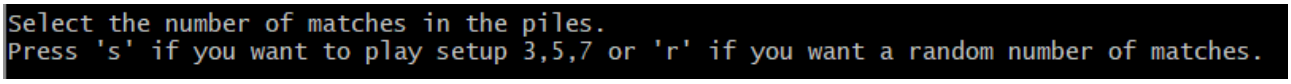
1. После запуска игры высвечивается инструкция. Чтобы начать игру, нужно нажать ‘s’.



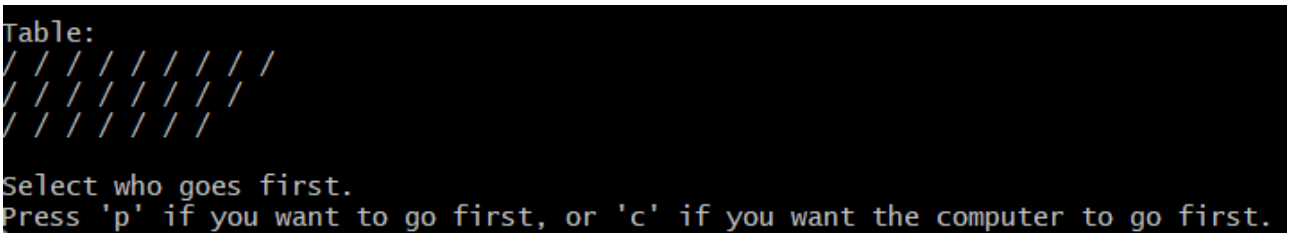
2. Далее необходимо выбрать уровень сложности.



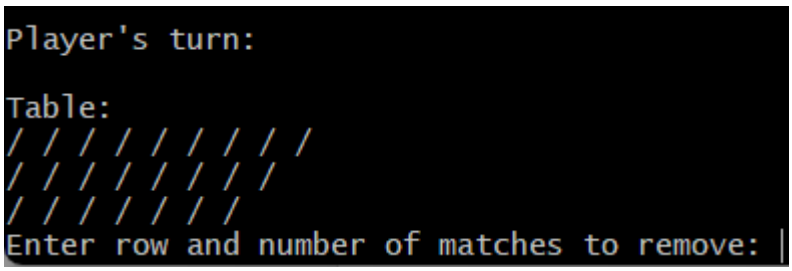
3. Затем нужно выбрать количество спичек в кучках (рандомное или статическое).



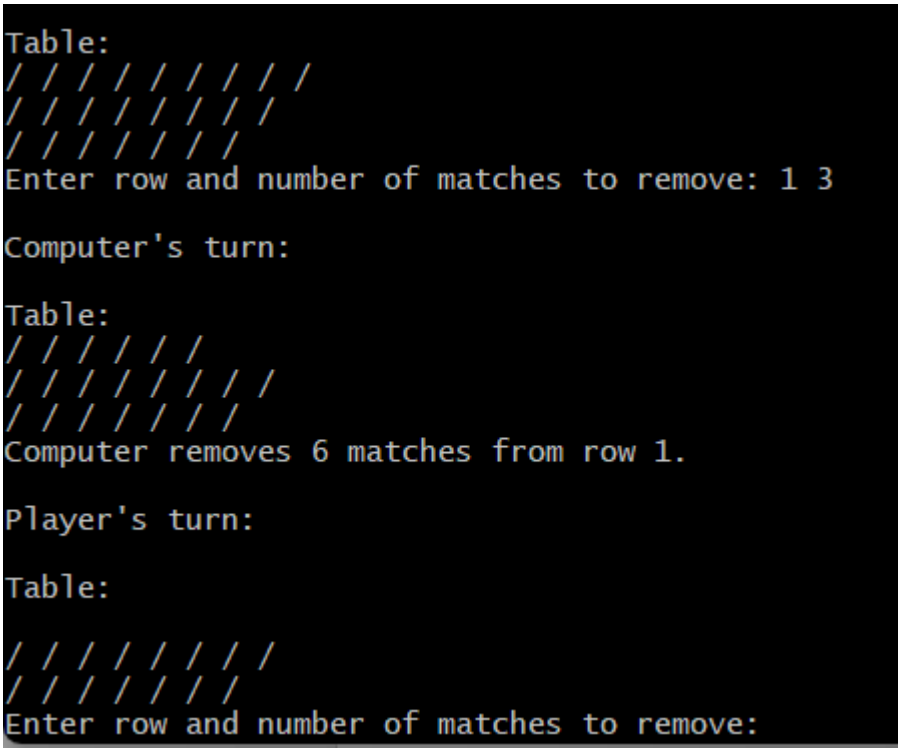
4. После этого высвечивается поле, и необходимо выбрать кто ходит первым.



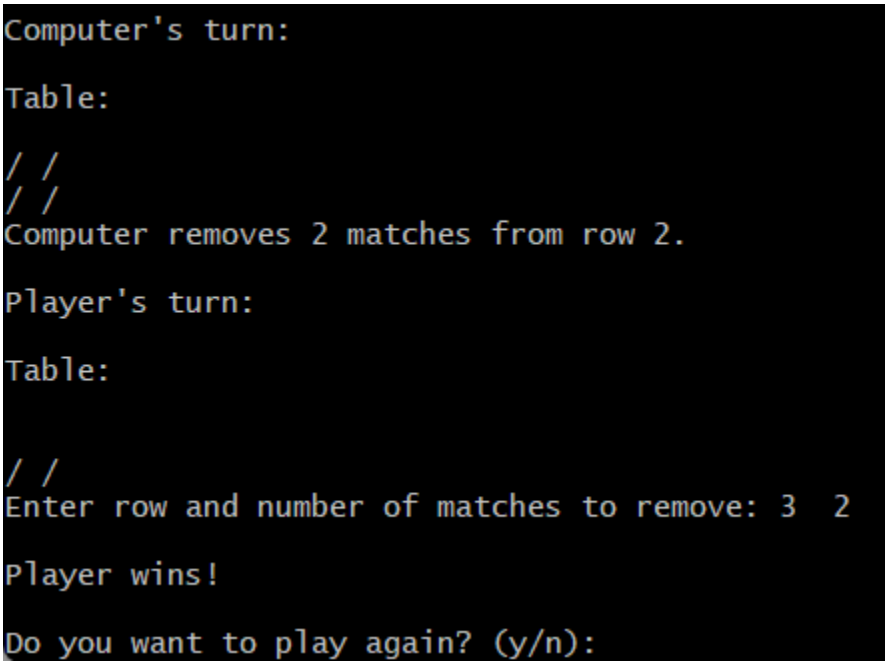
5. Если игрок ходит первым, то он выбирает кучку и количество спичек, которых нужно убрать.



6. Высвечивается поле после хода игрока, компьютер делает свой ход, и поле обновляется.



7. Игроки поочередно делают свой выбор. Игра продолжается пока кто-то не возьмет последние спички с поля. После выигрыша высвечивается кто победил.



8. Далее можно продолжить играть или выйти из игры.

Личный вклад в проект

В начале разработки компьютерной версии игры «Ним» передо мной стояла важная задача - создать структуры данных для представления игрового поля и игроков, а также написать функции для инициализации и вывода поля. Одним из ключевых моментов было обеспечение возможности отображения игрового поля с помощью символов, представляющих спички, и выбора пользователем статического или случайного расположения спичек.

Далее, в процессе разработки, была реализована функциональность выбора уровня сложности через консоль, а также была создана функция, позволяющая компьютеру делать случайные ходы. Кроме того, в ходе работы были внесены изменения в Makefile для оптимизации процесса сборки проекта.

Особое внимание уделялось проверки ввода данных пользователем, чтобы обеспечить корректность и надежность игрового процесса, предотвращая возможные ошибки и конфликты во время игры. Подготовленная инструкция предоставляла ясное объяснение правил игры, что способствовало более простому и приятному старту для игроков.

В заключительной стадии работы была добавлена функциональность игры по кругу, позволяющая сразу после завершения игры продолжить ее или выйти. Также были исправлены некоторые недочеты в коде, чтобы обеспечить более стабильную и удовлетворительную игровую среду.

Каждый этап разработки направлен на создание удобного и интуитивно понятного интерфейса, который обеспечивает надежность и стабильность приложения, а также приносит удовольствие игрокам в процессе игры.

Приложение. Текст программы

```
// file nim.cpp

#include <libnim/nim.hpp>
#include <stdio.h>
#include <stdlib.h>

void initializeStaticBoard(struct Board *board) {
     board->number_rows = MAX_ROWS;
    board->number_matches[0] = 3;
    board->number_matches[1] = 5;
    board->number_matches[2] = 7;
}

void initializeRandomBoard(struct Board *board) {
     board->number_rows = MAX_ROWS;
     board->number_matches[0] = rand() % 10 + 1;
     board->number_matches[1] = rand() % 10 + 1;
     board->number_matches[2] = rand() % 10 + 1;
}

void printBoard(struct Board *board) {
     printf("\nTable:\n");
     for (int i = 0; i < board->number_rows; i++) {
          for (int j = 0; j < board->number_matches[i]; j++) {
               printf("/ ");
          }
          printf("\n");
     }
}

void removeMatches(struct Board *board, int row, int number_matches) {
    if (board->number_matches[row-1] < number_matches) {
        printf("Invalid number of matches. Please try again.\n");
        return;
    }
    board->number_matches[row-1] -= number_matches;
}

int gameOver(struct Board *board) {
    for (int i = 0; i < board->number_rows; i++) {
        if (board->number_matches[i] > 0) {
            return 0;
        }
    }
    return 1;
}

void computerTurnWin(Board *board, int *row, int *number_matches, struct Player
*current_player) {
     int nim_sum = 0;
     for (int i = 0; i < board->number_rows; i++) {
         nim_sum ^= board->number_matches[i];
     }

     if (nim_sum == 0) {
          *row = rand() % MAX_ROWS + 1;
          *number_matches = rand() % (board->number_matches[*row - 1]) + 1;
     } else {
          for (int i = 0; i < board->number_rows; i++) {
```

```
                if ((board->number_matches[i] ^ nim_sum) < board-
>number_matches[i]) {
                    *row = i + 1;
                    *number_matches = board->number_matches[i] - (board-
>number_matches[i] ^ nim_sum);
                    break;
                }
            }
        }
        printf("%s removes %d matches from row %d.\n", current_player->name,
*number_matches, *row);
        removeMatches(board, *row, *number_matches);
}

void computerTurnRandom(Board *board, int *row, int *number_matches, struct
Player *current_player) {
        do {
            *row = rand() % board->number_rows;
        } while (board->number_matches[*row] == 0);
        *row += 1;

        *number_matches = rand() % board->number_matches[*row - 1] + 1;
        printf("%s removes %d matches from row %d.\n", current_player->name,
*number_matches, *row);
        removeMatches(board, *row, *number_matches);
}

void playGame(struct Board *board, struct Player *p1, struct Player *p2, char
difficulty_level) {
    struct Player *current_player = p1;
    int invalid_input;

    while (!gameOver(board)) {
        printf("\n%s's turn:\n", current_player->name);
        printBoard(board);
        int row, number_matches;
        if (current_player->id == 1) {
            do {
                invalid_input = 0;
                printf("Enter row and number of matches to remove: ");
                fflush(stdout);
                if (scanf("%d %d", &row, &number_matches) != 2) {
                    invalid_input = 1;
                    printf("Invalid input. Please enter two integers.\n");
                    fflush(stdout);
                    while (getchar() != '\n');
                    continue;
                }
                if (row < 1 || row > board->number_rows) {
                    invalid_input = 1;
                    printf("Invalid row. Please try again.\n");
                    fflush(stdout);
                    continue;
                }
                if (number_matches < 1 || number_matches > board-
>number_matches[row-1]) {
                    invalid_input = 1;
                    printf("Invalid number of matches. Please try again.\n");
                    fflush(stdout);
                    continue;
                }
                removeMatches(board, row, number_matches);
            } while (invalid_input);
```

```
        } else {
           if (difficulty_level == 'h') {
                    computerTurnWin(board, &row, &number_matches,
current_player);
           }
           else {
                computerTurnRandom(board, &row, &number_matches,
current_player);
                }
           }
        if (gameOver(board)) {
            printf("\n%s wins!\n", current_player->name);
            fflush(stdout);
        } else {
            current_player = (current_player == p1) ? p2 : p1;
        }
    }
}
```

```
// file nim.hpp
#pragma once

#define MAX_ROWS 3

struct Board {
    int number_rows;
    int number_matches[MAX_ROWS];
};

struct Player {
    char name[50];
    int id;
};

void initializeStaticBoard(struct Board *board);
void initializeRandomBoard(struct Board *board);
void printBoard(struct Board *board);
void removeMatches(struct Board *board, int row, int number_matches);
int gameOver(struct Board *board);
void computerTurnWin(Board *board, int *row, int *number_matches, struct Player
*current_player);
void playGame(struct Board *board, struct Player *p1, struct Player *p2, char
difficulty_level);
void computerTurnRandom(Board *board, int *row, int *number_matches, struct
Player *current_player);
```

```
//file main.cpp
#include <libnim/nim.hpp>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
      srand(time(NULL));
      printf("Hello! This is a NIM game!\nRules of the game:\n");
      printf("There will be three piles of matches in front of you.\nIn one
turn, you can take any number of matches, but only from one pile.\n");
      printf("The player who takes the last match wins.\n\n");

      char start;
      do {
            printf("Enter 's' to start ");
            fflush(stdout);
            scanf(" %c", &start);
            while (getchar() != '\n');
      } while (start != 's');

      char play_again;
      do {
            struct Board board;
            struct Player player = {"Player", 1};
            struct Player computer = {"Computer", 2};

            char difficulty_level;
            printf("\nSelect difficulty level\n");
            do {
                  printf("Press 'e' for easy difficulty or 'h' for hard ");
                  fflush(stdout);
                  scanf(" %c", &difficulty_level);
                  while (getchar() != '\n');
            } while (difficulty_level != 'e' && difficulty_level != 'h');

            char num_matches;
            printf("\nSelect the number of matches in the piles.\n");
            do {
                  printf("Press 's' if you want to play setup 3,5,7 or 'r' if you
want a random number of matches. ");
                  fflush(stdout);
                  scanf(" %c", &num_matches);
                  while (getchar() != '\n');
            } while (num_matches != 's' && num_matches != 'r');

            if (num_matches == 's') {
                  initializeStaticBoard(&board);
                  printBoard(&board);
            }
            if (num_matches == 'r') {
                  initializeRandomBoard(&board);
                  printBoard(&board);
            }

            char first_player;
            printf("\nSelect who goes first.\n");
           do {
               printf("Press 'p' if you want to go first, or 'c' if you want the
computer to go first. ");
               fflush(stdout);
               scanf(" %c", &first_player);
```

```
            while (getchar() != '\n');
        } while (first_player != 'p' && first_player != 'c');

         if (first_player == 'p') {
            playGame(&board, &player, &computer, difficulty_level);
        } else if (first_player == 'c') {
            playGame(&board, &computer, &player, difficulty_level);
        }

        printf("\nDo you want to play again? (y/n): ");
        fflush(stdout);
        scanf(" %c", &play_again);
        while (getchar() != '\n');
        if (play_again == 'y') {
           printf("\n");
       }
    } while (play_again == 'y');

    return 0;
}
```