



توضیحات پروژه ی مبانی برنامه نویسی کامپیوتر
(Dots And Boxes)

اسامی اعضای گروه پروژه:

محمد علی گلشن

سروش میرشکاری

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <windows.h>
```

```
using namespace std;
```

این خطوط، کتابخانه‌های مورد نیاز را شامل می‌شود. `iostream` برای ورودی و خروجی، `vector` برای استفاده از آرایه‌های پویا، `algorithm` برای الگوریتم‌های استاندارد، `fstream` برای فایل خوانی و فایل نویسی، `sstream` برای کار با رشته‌ها و `windows.h` برای دسترسی به توابع ویندوز.

```
string const SCORE_BOARD = "scoreboard.csv";
```

```
const int MAX_ROWS = 100;
```

```
const int MAX_COLS = 2;
```

ثابت‌های مورد نیاز تعریف می‌شوند. `SCORE_BOARD` نام فایل ذخیره امتیازات است. `MAX_ROWS` و `MAX_COLS` حداکثر تعداد سطرها و ستون‌ها را مشخص می‌کنند.

کلاسی به نام `ScoreBoardRow` تعریف می‌شود که شامل نام و تعداد بردهای یک بازیکن است. این کلاس عملگر `>` را بازتعریف می‌کند تا امتیازات به ترتیب نزولی مرتب شوند.

```
class ScoreBoardRow {
public:
    string name;
    int wins;
    ScoreBoardRow(string n, int w) {
```

```

        name = n;
        wins = w;
    }
    bool operator<(const ScoreBoardRow &obj) const {
        return wins > obj.wins;
    }
};

```

```

string left_number(int number, int width) {
    string num = to_string(number);
    int white_space = width - num.length();
    for (int i = 0; i < white_space; ++i) {
        num += " ";
    }
    return num;
}

```

تابعی برای قالب‌بندی اعداد به صورتی که همیشه تعداد مشخصی کاراکتر داشته باشند، با افزودن فاصله‌های خالی به انتهای آنها.

```

void screen_wipe();
void write_to_file(vector<ScoreBoardRow> v);
vector<ScoreBoardRow> read_score_board();
void score_board(bool wipe);
void main_menu(bool wipe);

```

تعریف اولیه چند تابع که بعداً پیاده‌سازی خواهند شد. این توابع مربوط به پاک کردن صفحه، نوشتن به فایل، خواندن از فایل، نمایش تابلو امتیازات و نمایش منوی اصلی هستند.

```
class Covertor {
public:
    char o;
    int y;
    int x;
    Covertor(int ix, int iy) {
        if (ix % 2 == 1) {
            o = 'h';
            x = (ix - 1) / 2;
            y = iy / 2;
        } else {
            o = 'v';
            x = ix / 2;
            y = (iy - 1) / 2;
        }
    }
};
```

کلاسی به نام Covertor که مختصات خطوط افقی و عمودی را مدیریت می‌کند. این کلاس از دو مقدار ورودی ix و iy استفاده کرده و نوع خط (افقی یا عمودی) و موقعیت آن را تعیین می‌کند.

```
class Player {
private:
```

```
    string name;
    int color;
    int score;
    int wins = 0;
public:
    Player (string n, int c, int s) {
        name = n;
        color = c;
        score = s;
    }
    string get_name() {
        return name;
    }
    int get_color() {
        return color;
    }
    void add_score(int s) {
        score += s;
    }
    int get_score() {
        return score;
    }
    void add_win() {
        wins++;
    }
```

```
void reset_score() {
```

```
    score = 0;
```

```
}
```

```
int get_wins() {
```

```
    return wins;
```

```
}
```

```
bool operator<(const Player &obj) const {
```

```
    return wins > obj.wins;
```

```
}};
```

کلاسی به نام Player که اطلاعات مربوط به یک بازیکن را شامل می‌شود. این کلاس شامل متدهایی برای گرفتن و تغییر نام، رنگ، امتیاز و تعداد بردها است. همچنین عملگر > برای مرتب‌سازی بازیکنان بر اساس تعداد بردها بازتعریف شده است.

```
vector<Player> all_players;
```

```
int height = 4;
```

```
int width = 4;
```

تعریف متغیرهای سراسری all_players. لیستی از تمام بازیکنان و height و width اندازه‌های پیش فرض بازی هستند.

```
void update_score_board(vector<Player> p);
```

```
void color_print(string s, int color);
```

```
void add_win(string name);
```

تعریف اولیه چند تابع دیگر که بعداً پیاده‌سازی خواهند شد. این توابع برای به‌روزرسانی تابلو امتیازات، چاپ رنگی متن و افزودن برد به بازیکن خاص استفاده می‌شوند.

```
class Game {
private:
    int total_boxes = 0;
    int game_height;
    int game_widht;
    vector<vector<int>> hlines;
    vector<vector<int>> vlines;
    vector<vector<int>> boxes;
    int turn = 0;
public:
    Game(int h, int w) {
        game_height = h;
        game_widht = w;
        for (int i = 0; i < game_height; ++i) {
            vector<int> t;
            for (int j = 0; j < game_widht - 1; ++j) {
                t.push_back(-1);
            }
            hlines.push_back(t);
        }
        for (int i = 0; i < game_height - 1; ++i) {
            vector<int> t;
```

```

        for (int j = 0; j < game_widht; ++j) {
            t.push_back(-1);
        }
        vlines.push_back(t);
    }
    for (int i = 0; i < game_height - 1; ++i) {
        vector<int> t;
        for (int j = 0; j < game_widht - 1; ++j) {
            t.push_back(-1);
        }
        boxes.push_back(t);
    }
}

```

کلاسی به نام Game که بازی را مدیریت می کند. در سازنده کلاس، بردهای افقی و عمودی و جعبه ها با مقدار اولیه ۱ - مقداردهی می شوند.

```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe
now is golshan turn

  0  1  2  3  4  5  6
0  *  *  *  *  *  *
1
2  *  *  *  *  *  *
3
4  *  *  *  *  *  *
5
6  *  *  *  *  *  *

soroosh : 0 | golshan : 0 |
please give line coordination and orientation : or enter -1 -1 to leave
0_

```

```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe
now is soroosh turn

  0  1  2  3  4  5  6
0  *  *  *  *  *  *
1
2  *  *  *  *  *  *
3
4  *  *  *  *  *  *
5
6  *  *  *  *  *  *

soroosh : 0 | golshan : 0 |
please give line coordination and orientation : or enter -1 -1 to leave
0_

```



```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe
1-soroosh 1
2-golshan 0
press 0 to return

```

```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe
now is soroosh turn

  0   1   2   3   4   5   6
0  *---*---*---*---*
1  |*****|*****|*****|
2  |*****|*****|*****|
3  *---*---*---*---*
4  |*****|*****|*****|
5  |*****|*****|*****|
6  *---*---*---*---*

soroosh : 8 | golshan : 1 |
game ended!
soroosh wins the game!
1-soroosh 1
2-golshan 0
press 0 to return

```

```

void draw_board(bool wipe) {
    if (wipe) { screen_wipe(); }

    cout << "    ";
    cout << "now is ";
    color_print(all_players[turn].get_name(), all_players[turn].get_color());
    cout << " turn" << endl << endl << endl;
    cout << "    ";
    for (int i = 0; i < widht * 2 - 1; ++i) {
        cout << left_number(i, 5);
    }
    cout << endl << endl;
    for (int row = 0; row < height; ++row) {
        cout << "    ";
        cout << left_number(row * 2, 5);
        for (int col = 0; col < widht - 1; ++col) {

```

```

cout << "*";
if (hlines[row][col] == -1) {
    cout << "    ";
} else {
    color_print("-----", all_players[hlines[row][col]].get_color());
}
}
cout << "*" << endl;

```

تابع `draw_board` برد بازی را رسم می‌کند. اگر `wipe` برابر `true` باشد، صفحه پاک می‌شود. سپس برد بازی بر اساس بردهای افقی، عمودی و جعبه‌ها رسم می‌شود. در این بخش، خطوط افقی و عمودی با استفاده از متغیرهای `hlines` و `vlines` و بر اساس نوبت بازیکنان به صورت رنگی نمایش داده می‌شوند.

```

if (row < height - 1) {
    cout << "    ";
    for (int col = 0; col < widht; ++col) {
        if (vlines[row][col] == -1) {
            cout << "    ";
        } else {
            if (boxes[row][col] == -1) {
                color_print("|", all_players[vlines[row][col]].get_color());
            } else {
                color_print("|", all_players[vlines[row][col]].get_color());
                if (col < game_widht - 1) {
                    color_print("*****", all_players[boxes[row][col]].get_color());
                }
            }
        }
    }
}

```

```

        }
    }
}
cout << endl;
cout << "    ";
cout << left_number(row * 2 + 1, 5);
for (int col = 0; col < widht; ++col) {
    if (vlines[row][col] == -1) {
        cout << "    ";
    } else {
        if (boxes[row][col] == -1) {
            color_print("|    ", all_players[vlines[row][col]].get_color());
        } else {
            color_print("|", all_players[vlines[row][col]].get_color());
            if (col < game_widht - 1) {
                color_print("*****", all_players[boxes[row][col]].get_color());
            }
        }
    }
}
cout << endl;
}

```

بخش آخر تابع draw_board همچنان به رسم برد بازی ادامه می دهد و خطوط عمودی و جعبه ها را نمایش می دهد. در ادامه این قسمت را به صورت کامل توضیح می دهیم:

این کد برای هر ردیف که کمتر از 1 - height است، خطوط عمودی را رسم می کند و اگر یک جعبه کامل شده باشد، آن را نیز نمایش می دهد.

ابتدا $\text{if (row < height - 1)}$ بررسی می کند که آیا ردیف فعلی کمتر از تعداد کل ردیف ها است یا نه. اگر بله، خطوط عمودی رسم می شوند.

$\text{for (int col = 0; col < width; ++col)}$ یک حلقه برای تمام ستون ها اجرا می کند.

$\text{if (vlines[row][col] == -1)}$ بررسی می کند که آیا خط عمودی در این مکان هنوز رسم نشده است یا نه.

اگر خط عمودی رسم نشده باشد، ۱۰ کاراکتر فاصله خالی (" ") چاپ می کند.

در غیر این صورت:

$\text{if (boxes[row][col] == -1)}$ بررسی می کند که آیا جعبه در این مکان کامل شده است یا نه. اگر جعبه کامل

نشده باشد، یک خط عمودی رنگی با فاصله (" |") چاپ می کند.

اگر جعبه کامل شده باشد، ابتدا خط عمودی رنگی ("|") چاپ می شود، سپس اگر ستون کمتر از game_width

۱- باشد، پنج ستاره رنگی ("*****") چاپ می کند.

پس از اتمام حلقه ستون ها، به خط بعدی می رود و یک شماره ردیف جدید و همچنین خطوط عمودی را برای

همان ردیف دوباره رسم می کند تا نمایش برد تکمیل شود.

تابع color_print نیز به این شکل عمل می کند که یک رشته را با رنگ مشخص شده چاپ می کند. برای مثال،

اگر بازیکنی با رنگ خاصی مشخص شده باشد، خطوط مربوط به او به همان رنگ نمایش داده می شوند.

در نهایت، کل این تابع وظیفه دارد تا برد بازی را به صورتی که بازیکنان بتوانند وضعیت فعلی بازی را ببینند، نمایش

دهد. این شامل خطوط افقی و عمودی، جعبه های کامل شده و نوبت بازیکن فعلی است.

توابع تکمیلی و اصلی بازی:

این توابع شامل توابع منو، مدیریت فایل، و موارد مرتبط با تغییرات و بروزرسانی های برد امتیازات هستند.

تابع player_menu

این تابع برای مدیریت منوی بازیکنان استفاده می‌شود:

```
void player_menu(bool wipe) {
    if (wipe) {
        screen_wipe();
    }
    int input;
    int del_input;
    string in_name;
    int in_color;
    cout << "color codes are: ";
    for (int i = 1; i < 15; ++i) {
        color_print(to_string(i), i);
        cout << " | ";
    }
    cout << endl;
    for (int i = 0; i < all_players.size(); ++i) {
        cout << "    " << i + 1 << " - ";
        color_print(all_players[i].get_name(), all_players[i].get_color());
        cout << endl;
    }
    cout << "enter: " << endl << "1 to add new player" << endl << "2 to delete player" << endl
        << "0 to return to main menu" << endl;
    cin >> input;
```

```

if (input == 0) {
    main_menu(true);
} else if (input == 1) {
    cout << "enter player name and color code to add it: " << endl;
    cin >> in_name >> in_color;
    if (cin.fail()) {
        cout << "you entered wrong input" << endl;
        cin.clear();
        cin.ignore(1000, '\n');
        player_menu(false);
    }
    Player temp(in_name, in_color, 0);
    all_players.push_back(temp);
    player_menu(true);
} else if (input == 2) {
    cout << "enter player number: " << endl;
    cin >> del_input;
    if (cin.fail()) {
        cout << "you entered wrong input" << endl;
        cin.clear();
        cin.ignore(1000, '\n');
        player_menu(false);
    }
    all_players.erase(all_players.begin() + del_input - 1);
    player_menu(false);
}

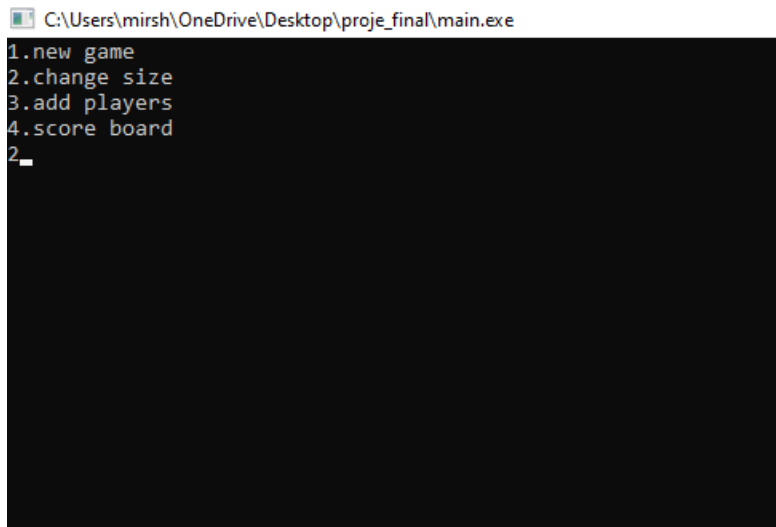
```

```

} else {
    cout << "wrong input!" << endl;
    cin.clear();
    cin.ignore(1000, '\n');
    player_menu(false);
}
}

```

این تابع برای مدیریت بازیکنان (افزودن، حذف، بازگشت به منوی اصلی) استفاده می شود.



```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe
1.new game
2.change size
3.add players
4.score board
2_

```

تابع score_board

این تابع برای نمایش و بروزرسانی برد امتیازات استفاده می شود:

```

void score_board(bool wipe) {
    if (wipe) {
        screen_wipe();
    }
    int x;

```

```

update_score_board(all_players);
vector<ScoreBoardRow> v = read_score_board();
sort(v.begin(), v.end());
for (int i = 0; i < v.size(); ++i) {
    cout << i + 1 << "-" << v[i].name << " " << v[i].wins << endl;
}
cout << "press 0 to return" << endl;
cin >> x;
if (x == 0) {
    main_menu(true);
} else {
    cout << "wrong input!" << endl;
    cin.clear();
    cin.ignore(1000, '\n');
    score_board(false);
}
}

```

این تابع ابتدا برد امتیازات را بروزرسانی می کند، سپس آن را می خواند و نمایش می دهد. در نهایت کاربر می تواند با فشردن عدد ۰ به منوی اصلی برگردد.

تابع read_score_board

این تابع برد امتیازات را از فایل scoreboard.csv می خواند:

```

vector<ScoreBoardRow> read_score_board() {

```



```

vector<ScoreBoardRow> rows;
ifstream file(SCORE_BOARD);
if (!file.is_open()) {
    cerr << "Error opening file!" << endl;
}
string data[MAX_ROWS][MAX_COLS];
string line;
int row = 0;
while (getline(file, line))
{
    if(line.length() == 0){
        break;
    }
    stringstream ss(line);
    string cell;
    int col = 0;
    while (getline(ss, cell, ',') && col < MAX_COLS) {
        data[row][col] = cell;
        col++;
    }
    row++;
}
file.close();
for (int i = 0; i < row; ++i) {
    rows.push_back(ScoreBoardRow(data[i][0], stoi(data[i][1])));
}

```

```

    }
    return rows;
}

```

این تابع خطوط فایل را می‌خواند و داده‌ها را در یک آرایه دو بعدی ذخیره می‌کند. سپس این داده‌ها را به برد امتیازات (یک vector از نوع ScoreBoardRow) تبدیل می‌کند.

تابع update_score_board

این تابع برای بروزرسانی برد امتیازات با داده‌های جدید بازیکنان استفاده می‌شود:

```

void update_score_board(vector<Player> p) {
    vector<ScoreBoardRow> v = read_score_board();
    vector<Player> missing;
    for (int i = 0; i < p.size(); ++i) {
        bool found = false;
        for (int j = 0; j < v.size(); ++j) {
            if (p[i].get_name() == v[j].name) {
                found = true;
            }
        }
        if (!found) {
            missing.push_back(p[i]);
        }
    }
    for (int i = 0; i < missing.size(); ++i) {
        v.push_back(ScoreBoardRow(missing[i].get_name(), missing[i].get_wins()));
    }
}

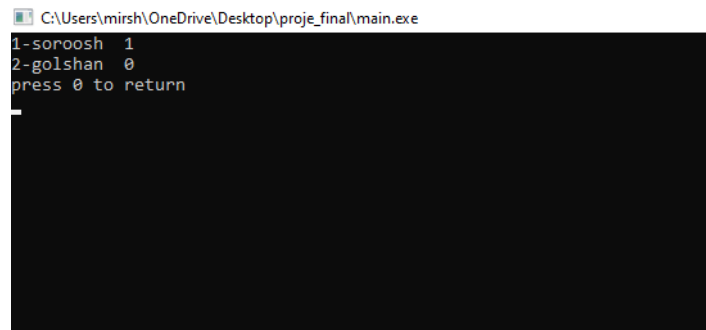
```

```

}
write_to_file(v);
}

```

این تابع بازیکنانی که در برد امتیازات نیستند را به آن اضافه می کند و سپس برد امتیازات را به روز می کند.



تابع add_win

این تابع تعداد بردهای بازیکنی که برنده شده را افزایش می دهد:

```

void add_win(string name) {
    vector<ScoreBoardRow> v = read_score_board();
    for (int i = 0; i < v.size(); ++i) {
        if (name == v[i].name) {
            v[i].wins++;
            break;
        }
    }
    write_to_file(v);
}

```

این تابع بردهای بازیکن مشخص شده را افزایش می دهد و تغییرات را به فایل ذخیره می کند.

تابع write_to_file

این تابع برد امتیازات را به فایل می‌نویسد:

```
void write_to_file(vector<ScoreBoardRow> v) {
    std::ofstream ofs;
    ofs.open(SCORE_BOARD, std::ofstream::out | std::ofstream::trunc);
    ofs.close();
    fstream fout;
    fout.open(SCORE_BOARD, ios::out | ios::app);
    for (int i = 0; i < v.size(); ++i) {
        fout << v[i].name << ", " << v[i].wins << "\n";
    }
    fout.close();
}
```

این تابع ابتدا فایل را خالی می‌کند و سپس برد امتیازات به‌روز شده را در آن می‌نویسد.

تابع اصلی main

تابع اصلی برنامه که منوی اصلی را فراخوانی می‌کند:

```
int main() {
    main_menu(true);
    return 0;
}
```

این تابع منوی اصلی را اجرا می‌کند که به کاربر امکان شروع بازی، تغییر ساینز برد، افزودن بازیکن، و مشاهده برد امتیازات را می‌دهد.

C:\Users\mirsh\OneDrive\Desktop\projc_final\main.exe

```
1.new game
2.change size
3.add ploycrs
4.score board
_
```

C:\Users\mirsh\OneDrive\Desktop\projc_final\main.exe

```
1.new game
2.change size
3.add players
4.score board
2_
```

C:\Users\mirsh\OneDrive\Desktop\projc_final\main.exe

```
current board size is height: 4 widht: 4
1.change
0.return to main menu
1_
```

C:\Users\mirsh\OneDrive\Desktop\projc_final\main.exe

```
current board size is height: 4 widht: 4
1.change
0.return to main menu
1
enter height and width:
5
5
```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe

```
1.new game
2.change size
3.add players
4.score board
3_
```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe

```
color codes are: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
enter:
1 to add new player
2 to delete player
0 to return to main menu
1
enter player name and color code to add it:
soroosh
5_
```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe

```
1.new game
2.change size
3.add players
4.score board
4_
```

C:\Users\mirsh\OneDrive\Desktop\proje_final\main.exe

```
1-soroosh 0
2-golshan 0
press 0 to return
_
```