Laboratorio di Linguaggi di Sistema

a.a. 2007/2008

Progetto finale III (base)

Autore: Federico Viscomi.

Sommario

1 Specifiche e scelte implementative

Struttura del codice e programma principale

Il progetto consta di quattro parti:

- *board.c – board.h*
- *playersManager.c – playersManager.h*
- *ioInterface.c – ioInterface.h*
- *game.c*

game.c è il file principale che contiene il metodo *main.* Quest'ultimo analizza gli argomenti passati al programma, inizializza un certo insieme di strutture dati e avvia la simulazione del gioco.
Conviene creare dei tipi astratti a parte per il tabellone e per i giocatori se vogliamo riunire le operazioni di input/output in un unica interfaccia. Avere un interfaccia di I/O separata dal resto del programma facilita la riusabilità del codice ad esempio nel caso in cui vogliamo scrivere un interfaccia grafica.

board.c

Implementa il tipo di dato astratto che rappresenta il tabellone del gioco.
Tale tabellone deve:

- mantenere le associazioni tra numero di casella e descrizione della prova relativa a quella casella
- permettere di aggiungere un certo numero di associazioni numero_casella-descrizione_prova
- reperire la descrizione della prova associata a una data casella usando il numero della casella come chiave di ricerca

Si deve implementare una collezione di elementi a cui sono associate delle chiavi prese da un insieme totalmente ordinato sulla quale sono definite le operazione di inserimento e ricerca. Dunque usiamo un dizionario. Scegliamo una struttura dati tra alcune possibili tenendo presente che:

- il tipo di dato non ha l'operazione di cancellazione o eventuali altre operazioni diverse da inserimento e ricerca
- possiamo fare ipotesi sulla dimensione massima del numero di caselle e quindi sulla dimensione massima dell'insieme delle chiavi
- i numeri delle caselle(e quindi le chiavi di ricerca) sono interi maggiori di 0
- se viene usata una chive in un operazione di inserimento allora verranno usate anche le precedenti
- possiamo impedire che vengano inserite associazioni con due chiavi uguali percheè se dovesse accadere sarebbe un errore di scrittura del file di configurazione
- il programma fa un certo numero di operazioni di inserimento prima dell'avvio del gioco seguito da un certo numero di operazioni di ricerca durante lo svolgimento del gioco.

Alla luce di queste considerazioni tra le varie strutture dati che si possono usare per implementare un dizionario conviene usare un array ordinato di dimensione fissa indicizzato col numero della casella.

PlayersManager.c

Implementa un tipo di dato che si occupa di gestire tutte le operazione che interessano i giocatori:

- creare e mantenere l'insieme dei giocatori con i relativi attributi: nome, casella in cui si trova al turno corrente, stato e altri.
- Leggere e modificare gli attributi dei giocatori. In particolare i giocatori possono essere il risultato di una ricerca in base a una chiave. Si deve anche poter iterare su tutta la collezzione dei giocatori

Abbiamo bisogno di un tipo di dato che sia una collezzione di elementi e che definisca le operazioni di inserimento e ricerca.
Considerazioni:

- il numero e il nome dei giocatori sono noti a tempo di esecuzione ed sono costanti per tutta la vita del programma
- il programma non deve terminare durante lo svolgimento del gioco. Dunque bisogna cercare di allocare la memoria necessaria prima dell'inizio del gioco.
- alla fine del gioco si deve avere una classifica dei giocatori

La struttura dati scelta è un array di puntatori a struct. Abbiamo una struct per ogni giocatore. Esaminiamo la complessità delle operazioni:

- la creazione ha complessità in tempo e spazio lineare nel numero di giocatori.
- la ricerca di un giocatore è costante in tempo e spazio. Sarebbe lineare nel numero di giocatori se si usasse come chiave il nome del giocatore. Imponendo che l'interfaccia di I/O visualizzi ad ogni turno la corrispondenza tra giocatore e numero del giocatore possiamo usare tale numero per indicizzare l'array ottenendo dunque una complessità costante. Come effetto collaterale abbiamo che il metodo dell'interfaccia che mostra lo stato del gioco ad ogni iterazione abbia complessità almeno lineare nel numero dei giocatori perchè per ogni giocatore deve mostrare il numero associato.
- la complessità in tempo e spazio della creazione della classifica è pari a quella del quick sort che si trova nell libreria standard c. Per ogni giocatore vengono memorizzati il turno in cui ha vinto e la casella in cui si trova. Per ottenere una classifica basta ordinare l'array dei giocatori secondo un certo criterio. Se due giocatori vincono in turni diversi allora nella classifica viene prima chi ha vinto nel turno minore; altrimenti viene prima chi è ha il numero di casella maggiore; altrimenti si considera pari.

ioInterface.c

Tutti i messagi di errore vengono visualizzati sullo stderr e memorizzati nel file di log. Gli errori riportati nel file di log sono tutti e soli quelli che si verificano dopo l'analisi degli argomenti e l'apertura del file stesso.
Tutto il resto dell'output viene indirizzato su stdout e l'input

viene letto da stdin con una semplice interfaccia testuale. Sia output che input vengono anche scritti sul file di log con la convenzione che il carattere '<' descrive l'input e '>' descrive l'output.

Formato del file di configurazione

Il file di configurazione ha il seguente formato:

```
numero_casella : descrizione\n
```

non sono ammesse linee vuote, ci deve essere un'associazione per ogni linea, il numero della casella deve essere maggiore di 0 inoltre non vengono accettate due associazioni distinte ma con il numero di casella uguale non per motivi di implementazione ma perchè potrebbe essere segno di errore nella scrittura del file di configurazione.

2 Codice

tutto il codice è stato indentato con il comando

```
indent -kr -br -brs -lc69 -l69 -nut *.c *.h
```

game.c

```c
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

#include "board.h"
#include "playersManager.h"
#include "ioInterface.h"

#define USAGE "game CONFIGURATION_FILE_NAME\
                LOG_FILE_NAME [PLAYER_NAME; 1..*]"
#define BUFF_SIZE (1 << 11)

int main(int argc, char *argv[])
{

    char **playersNameList;
    int totalPlayersNumber;
    int turn, i;
    char buffer[BUFF_SIZE];
    FILE *configFile;

    /* legge gli argomenti */
    char **name1, **name2;

    if (argc < 4) {
        fprintf(stderr,
                "ERROR: Wrong number of arguments. \n USAGE: %s\n",
                USAGE);
        exit(EXIT_FAILURE);
    }
    playersNameList = argv + 3;
    totalPlayersNumber = argc - 3;

    /* controlla se ci sono due giocatori con lo stesso nome */
    for (name1 = playersNameList; *name1; name1++)
        for (name2 = name1 + 1; *name2; name2++)
            if (strcmp(*name1, *name2) == 0) {
                fprintf(stderr, "ERROR: found two player with the"
                        "same name \"%s\"\n", *name1);
                exit(EXIT_FAILURE);
            }
    initIoInterface(argv[2]);
    /* crea e inizializza le strutture dati per i giocatori */
    initPlayersManager(totalPlayersNumber);
    for (; *playersNameList; playersNameList++)
        addPlayer(*playersNameList);
```

```
initBoard();
/*
 * legge il file di configurazione secondo il formato:
 *     numero_casella:descrizione della prova\n
 * e aggiunge le descrizioni al tabellone
 */
if ((configFile = fopen(argv[1], "r")) == NULL) {
    printErr("ERROR: error while opening configuration file\n");
    exit(EXIT_FAILURE);
}
while (fgets(buffer, BUFF_SIZE, configFile)) {
    char *description;
    int boxNumber;
    /* legge il numero di casella */
    if ((boxNumber = atoi(buffer)) <= 0) {
        printErr("ERROR:invalid box num(\"%s\") in"
                 " configuration file\n", buffer);
        exit(EXIT_FAILURE);
    }
    /* aggiunge una nuova casella con la relativa descrizione */
    if ((description = strchr(buffer, ':')) == NULL) {
        printErr("ERROR: missing ':' in configuration file\n");
        exit(EXIT_FAILURE);
    }
    addBox(boxNumber, description + 1);
}
if (getTotalBoxesNumber() == 0) {
    printErr("ERROR: invalid configuration file\n");
    exit(EXIT_FAILURE);
}
fclose(configFile);
printBoard();
showGame();
/* avvia la simulazione del gioco */
srand(time(NULL));
for (turn = 0; !allPlayersDone(); turn++) {
    if (!nextStep())
        return EXIT_SUCCESS;
    printMessage("\n**************************************\n");
    printMessage("turno %d", turn + 1);
    printMessage("\n**************************************\n");
    showGame();
    /*
     * per ogni giocatore G che non ha terminato il gioco:
     * 1. se G  e' fermo per un turno cambia il suo stato in
     *      modo che al turno successivo venga rimesso in gioco
     * 2. altrimenti viene lanciato il dado, mosso il giocatore
     *              e visualizzata la sua prova
     */
    while (nextPlayer()) {
        int state = getPlayerState();

        if (state == ACTIVE || state == TEST_PASSED
            || state == TO_BE_ACTIVATED) {
            if (state != ACTIVE)
                setPlayerState(ACTIVE, 0);
```

```
                movePlayer((rand() % 6) + 1);
                if (getPlayerBox() > getTotalBoxesNumber())
                    setPlayerState(DONE, turn);
                else
                    printMessage("player %s: \"%s\"\n",
                                 getPlayerName(),
                                 getDescription(getPlayerBox()));
            } else if (state == OUT_OF_TURN)
                setPlayerState(TO_BE_ACTIVATED, 0);
        }
        showGame();
        /*
         * Legge e registra l'esito di tutte le prove sostenute nel
         * turno corrente dai giocatori
         */
        for (i = getActivePlayersNumber(); i > 0; i--) {
            int playerNumber;
            bool result;

            do {
                result = askPlayerResult(&playerNumber);
                if (playerNumber > totalPlayersNumber)
                    printErr("WARNING: player number %d out of "
                             "bounds [1; %d]\n", playerNumber,
                             totalPlayersNumber);
                else {
                    setCurrentPlayer(playerNumber);
                    if (getPlayerState() != ACTIVE)
                        printErr("WARNING: player number %d not "
                                 "valid because player:"
                                 "\n\t-won"
                                 "\n\t-is out of turn"
                                 "\n\t-already passed the test\n",
                                 playerNumber);
                }
            }
            while (playerNumber > totalPlayersNumber
                   || getPlayerState() != ACTIVE);
            if (result)
                setPlayerState(TEST_PASSED, 0);
            else
                setPlayerState(OUT_OF_TURN, 0);
        }
    }
    printScore();
    closeIoInterface();

    return EXIT_SUCCESS;
}
```

board.h

```c
#ifndef BOARD_H_
#define BOARD_H_

/*
 * Inizializza le strutture dati che rappresentano il tabellone
 * del gioco. Quest'ultimo e' una funzione parziale:
 *              T : {numeri naturali} ---> {stringhe}
 * dove
 *              T(i) = s
 * se e solo se la casella i del tabellone ha come prova quella
 * descritta da s
 */
void initBoard();

/*
 * aggiunge al tabellone l'associazione: casella numero boxNumber,
 * descrizione description se questa non era gia presente altrimenti
 * termina l'esecuzione del programma informando l'utente con un
 * messagio d'errore.
 * In altri termini:
 *          se (boxNumber, description) non appartiene a T
 *          allora T diventa T U {(boxNumber, description)}
 */
void addBox(int boxNumber, char *description);

/*
 * cerca nel tabellone la descrizione della casella numero boxNumber
 * Restituisce tale descrizione in caso di successo altrimenti
 * termina l'esecuzione del programma informando  l'utente con un
 * messagio d'errore.
 * In altri termini:
 *    se c'e' una stringa s tale che (boxNumber, s) appartiene a T
 *    allora restituisce s
 *    altrimenti termina l'esecuzione del programma informando
 *                l'utente con un messagio d'errore.
 */
char *getDescription(int boxNumber);

/*
 * Restituisce il numero totale di caselle contenute nel tabellone.
 */
int getTotalBoxesNumber();

#endif                          /* BOARD_H_ */
```

board.c

```c
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "board.h"
#include "ioInterface.h"

#define MAX_BOXES_NUMBER 512

static char *descriptions[MAX_BOXES_NUMBER];
static int totalBoxesNumber;


void initBoard()
{
    memset(descriptions, '\0', sizeof(descriptions));
    totalBoxesNumber = 0;
}

void addBox(int boxNumber, char *description)
{
    int len;

    if (boxNumber > totalBoxesNumber)
        totalBoxesNumber = boxNumber;
    boxNumber--;
    if (boxNumber < 0 || boxNumber >= MAX_BOXES_NUMBER) {
        printErr("box number %d out of bounds [0; %d]\n", boxNumber,
                 MAX_BOXES_NUMBER - 1);
        exit(EXIT_FAILURE);
    }
    for (description++; isspace(*description); description++);
    if ((len = strlen(description)) == 0) {
        printErr("WARINING: empty description\n");
        return;
    }
    if (descriptions[boxNumber] != NULL) {
        printErr("readding description for box number %d\n",
                 boxNumber);
        exit(EXIT_FAILURE);
    }
    if ((descriptions[boxNumber] =
         (char *) malloc(sizeof(char) * (len + 1))) == NULL) {
        printErr("unable to allocate memory for a descritpion\n");
        exit(EXIT_FAILURE);
    }
    strcpy(descriptions[boxNumber], description);
}


char *getDescription(int boxNumber)
{
    if (boxNumber <= 0 || boxNumber > MAX_BOXES_NUMBER) {
        printErr("box number out of index\n");
        exit(EXIT_FAILURE);
    }
```

```
    return descriptions[boxNumber - 1] ? descriptions[boxNumber -
                                                      1] :
        "ATTENZIONE: nessuna descrizione associata alla casella\n";
}

int getTotalBoxesNumber()
{
    return totalBoxesNumber;
}
```

playersManager.h

```c
#ifndef PLAYERSMANAGER_H_
#define PLAYERSMANAGER_H_

#include <stdbool.h>

#define ACTIVE 0           /* il concorrente e' correntemente attivo
                            * e quindi partecipa al gioco */
#define DONE 1             /* il concorrente ha terminato il gioco */
#define OUT_OF_TURN 2      /* il concorrente deve saltare il turno
                            * corrente per cui riprendera' a
                            * partecipare al gico al turno successivo
                            */
#define TO_BE_ACTIVATED 3  /* il concorrente ha saltato il turno
                            * precedente dunque gioca nel turno
                            * corrente */
#define TEST_PASSED 4      /* il concorrente ha superato la prova */

/*
 * inizializza il gestore dei giocatori
 */
void initPlayersManager();

/*
 * agggiunge un nuovo giocatore nell'insieme dei giocatori
 */
void addPlayer(char *newPlayerName);

/*
 * fornisce un iteratore sulla collezzione dei giocatori.
 * restutuisce true se e' presente un giocatore non ancora
 * considerato e imposta tale giocatore come parametro implicito dei
 * metodi che agiscono su uno specifico giocatore cioe':
 *    void movePlayer(int dice); char *getPlayerName();
 *    char *getPlayerStateName(); int getState();
 *    int getPlayerBox(); int getPlayerTurn();
 *    int getPlayerWinning();  int getPlayerFailure();
 *    void setState(int state, int winTurn);
 * se tutti i giocatori sono gia stati usati allora restituisce
 * false e agisce in modo che successive chiamate a nextPlayer
 * permettano di esaminare nuovamente tutti i giocatori una e una
 * sola volta.
 */
bool nextPlayer();

/*
 * muove il giocatore corrente di un numero di casella pari al
 * parametro
 */
void movePlayer(int dice);

/*
 * restituisce true se tutti i giocatori hanno terminato il gioco;
 * false altrimenti
 */
bool allPlayersDone();

/*
```

```
 * restituisce il nome del giocatore corrente
 */
char *getPlayerName();

/*
 * restituisce una stringa che rappresenta lo stato del giocatore
 * corrente
 */
char *getPlayerStateName();

/*
 * restituisce il numero di giocatori attivi
 */
int getActivePlayersNumber();

/*
 * restituisce lo stato del giocatore corrente
 */
int getPlayerState();

/*
 * restituisce la casella del giocatore corrente
 */
int getPlayerBox();

/*
 * restituisce il turno di vincita del giocatore corrente
 */
int getPlayerTurn();

/*
 * restituisce il numero di prove vinte del giocatore corrente
 */
int getPlayerWinning();

/*
 * restituisce il numero di prove perse del giocatore corrente
 */
int getPlayerFailure();

/*
 * imposta lo stato del giocatore corrente a state e in caso il
 * giocatore abbia terminato il gioco memorizza il turno di vincita
 */
void setPlayerState(int state, int winTurn);

/*
 * imposta il giocatore corrente a quello numero currentPlayer
 */
void setCurrentPlayer(int currentPlayer);


#endif                          /* PLAYERSMANAGER_H_ */
```

playersManager.c

```c
#include <stdlib.h>
#include <string.h>

#include "playersManager.h"
#include "board.h"
#include "ioInterface.h"

static char *states[5] = {
    "attivo",
    "terminato",
    "fermo un turno",
    "da attivare",
    "prova superata"
};

/* struttura che rappresenta un singolo giocatore */
struct player_t {
    /* nome del giocatore */
    char *name;
    /* caselle in cui si trova il giocatore al turno corrente */
    int box;
    /* stato del giocatore */
    unsigned char state;
    /* turno in cui il giocatore termina il gioco */
    int turn;
    /* numero di prove superate */
    int won;
    /* numero di prove perse */
    int lost;
};


/* giocatori che non hanno terminato il gioco al turno corrente */
static int unDonePlayers;
static int totalPlayersNumber;
/*
 * indice del giocatore corrente che fa da parametro implicito
 * di metodi che lavorano su un giocatore specifico. questo
 * valore viene usato nal metodo nextPlayer per implementare un
 * iteratore su tutta la collezzione dei giocatori
*/
static int current;
static struct player_t **playersArray;

void initPlayersManager(int num)
{
    if ((playersArray =
         (struct player_t **) calloc(num,
                                     sizeof(struct player_t *))) ==
        NULL) {
        printErr("unable to allocate memory for players\n");
        exit(EXIT_FAILURE);
    }
    totalPlayersNumber = num;
    current = -1;
    unDonePlayers = 0;
}
```

```
void addPlayer(char *newPlayerName)
{
    struct player_t *newPlayer;
    if ((newPlayer =
         (struct player_t *) calloc(1,
                                    sizeof(struct player_t))) ==
        NULL) {
        printErr("unable to allocate memory for a new player");
        exit(EXIT_FAILURE);
    }
    newPlayer->name = newPlayerName;
    newPlayer->state = ACTIVE;
    playersArray[unDonePlayers] = newPlayer;
    unDonePlayers++;
}

bool nextPlayer()
{
    current++;
    if (current == totalPlayersNumber) {
        current = -1;
        return false;
    }
    return true;
}

int getActivePlayersNumber()
{
    int activePlayer, i;

    for (activePlayer = 0, i = 0; i < totalPlayersNumber; i++)
        if (playersArray[i]->state == ACTIVE)
            activePlayer++;
    return activePlayer;
}

static int comparePlayers(const void *arg1, const void *arg2)
{
    struct player_t *player1 = *((struct player_t **) arg1);
    struct player_t *player2 = *((struct player_t **) arg2);
    if (player1->turn != player2->turn)
        return player1->turn - player2->turn;
    return player2->box - player1->box;
}

void setPlayerState(int state, int turn)
{
    playersArray[current]->state = state;
    if (state == OUT_OF_TURN)
        playersArray[current]->lost++;
    else if (state == TEST_PASSED)
        playersArray[current]->won++;
    else if (state == DONE) {
        unDonePlayers--;
        playersArray[current]->turn = turn;
```

```
        if (unDonePlayers == 0)
            qsort(playersArray, totalPlayersNumber,
                  sizeof(struct player_t *), comparePlayers);
    }
}

int getPlayerState()
{
    return playersArray[current]->state;
}
void setCurrentPlayer(int currentPlayer)
{
    current = currentPlayer - 1;
}
void movePlayer(int dice)
{
    playersArray[current]->box += dice;
}

int getPlayerBox()
{
    return playersArray[current]->box;
}

char *getPlayerName()
{
    return playersArray[current]->name;
}

int getPlayerTurn()
{
    return playersArray[current]->turn;
}

int getPlayerWinning()
{
    return playersArray[current]->won;
}

int getPlayerFailure()
{
    return playersArray[current]->lost;
}

bool allPlayersDone()
{
    return unDonePlayers == 0;
}

char *getPlayerStateName()
{
    return states[playersArray[current]->state];
}
```

ioInterface.h

```c
#ifndef IOINTERFACE_H_
#define IOINTERFACE_H_

#include <stdbool.h>
#include <stdio.h>

#include "playersManager.h"
#include "board.h"

/*
 * Chiede all'utente il numero di un giocatore (> 0) e l'esito della
 * sua prova. memorizza tale numero nel valore puntato dal parametro
 * playerNumber e restituisce true in caso il giocatore abbia
 * superato la prova, false altrimenti. prima di chiedere il numero
 * di un giocatore l'interfaccia deve assicurare di mostrare quale
 * numero corrisponde a quale giocatore.
 */
bool askPlayerResult(int *playerNumber);

/*
 * chiede conferma di prosugire con il turno successivo
 */
bool nextStep();

/*
 * visualizza una descrizione dello stato del gioco
 */
void showGame();

/*
 * visualizza la classifica
 */
void printScore();

/*
 * visualizza un messaggio d'errore secondo il formato format
 */
void printErr(const char *format, ...);

/*
 * visualizza un messaggio secondo il formato format
 */
void printMessage(const char *format, ...);

/*
 * visualizza una descrizione del tabellone
 */
void printBoard();

/*
 * chiude l'interfaccia di I/O
 */
void closeIoInterface();

/*
 * inizializza l'interfaccia di I/O
 */
```

```
void initIoInterface(char *logFileName);

#endif                          /* IOINTERFACE_H_ */
```

ioInterface.c

```
#include <stdarg.h>
#include <string.h>
```

```c
#include <stdlib.h>

#include "ioInterface.h"

#define LINE_LEN (NUMBER_LEN + NAME_LEN + BOX_LEN + STATE_LEN + 13)
#define NUMBER_LEN 6
#define NAME_LEN 15
#define BOX_LEN 5
#define STATE_LEN 14
#define POS_LEN 10
#define WIN_LEN 14
#define FAIL_LEN 12
#define SCORE_LINE_LENGTH (POS_LEN + WIN_LEN + \
                                        FAIL_LEN + NAME_LEN + 13)
#define PRINT_LINE(character, output, line_length){\
      int i;\
      fprintf(output, "> ");\
      for (i = 0; i < line_length; i++)\
            fprintf(output, "%c", character);\
      fputc('\n', output);\
}

static FILE *logFile;

static void printGameStatusToFileP(FILE * out)
{
    int i = 1;
    PRINT_LINE('_', out, LINE_LEN);
    fprintf(out, "> | %*s | %*s | %*s | %*s |\n",
            NUMBER_LEN, "number", NAME_LEN, "name", BOX_LEN,
            "box", STATE_LEN, "state");
    PRINT_LINE('-', out, LINE_LEN);
    while (nextPlayer())
        fprintf(out,
                "> | %*d | %*s | %*d | %*s |\n",
                NUMBER_LEN, i++, NAME_LEN, getPlayerName(), BOX_LEN,
                getPlayerBox(), STATE_LEN, getPlayerStateName());
    PRINT_LINE('-', out, LINE_LEN);
}

static void printScoreToFileP(FILE * out)
{
    int position;
    int previous_box;
    int previous_turn;

    PRINT_LINE('_', out, SCORE_LINE_LENGTH);
    fprintf(out, "> | %*s | %*s | %*s | %*s |\n", POS_LEN,
            "posizione", NAME_LEN, "name", WIN_LEN,
            "prove superate", FAIL_LEN, "non superate");
    PRINT_LINE('-', out, SCORE_LINE_LENGTH);

    position = 1;
    previous_box = -1;
    previous_turn = -1;
    while (nextPlayer()) {
        fprintf(out, "> | %*d | %*s | %*d | %*d |\n", POS_LEN,
```

```
                position, NAME_LEN, getPlayerName(), WIN_LEN,
                getPlayerWinning(), FAIL_LEN, getPlayerFailure());
        if (getPlayerBox() != previous_box)
            position++;
        else if (getPlayerTurn() != previous_turn)
            position++;
        previous_box = getPlayerBox();
        previous_turn = getPlayerTurn();
    }
    PRINT_LINE('-', out, SCORE_LINE_LENGTH);
}


void printScore()
{
    printMessage("\n\n\nCLASSIFICA: \n\n");
    printScoreToFileP(logFile);
    printScoreToFileP(stdout);
}

void showGame()
{
    printGameStatusToFileP(logFile);
    printGameStatusToFileP(stdout);
}

bool askPlayerResult(int *playerNumber)
{
    char answer;

    do {
        printf("numero del giocatore seguito dall'esito "
               "della prova Y/N: \n");
        while (scanf("%d %c", playerNumber, &answer) != 2)
            scanf("%*[^\n]");
        if (*playerNumber <= 0 || strchr("YyNn", answer) == NULL)
            printf("valori inseriti non validi\n");
    }
    while (*playerNumber <= 0 || strchr("YyNn", answer) == NULL);
    putchar('\n');
    fprintf(logFile, "> inserire il numero del giocatore e"
            " l'esito della prova \n");
    fprintf(logFile, "< giocatore numero %d esito della prova %s\n",
            *playerNumber, (answer == 'Y'
                            || answer ==
                            'y') ? "SUPERATA" : "NON SUPERATA");
    return (answer == 'Y' || answer == 'y');
}

bool nextStep()
{
    char answer;

    do {
        fprintf(stdout, "next step?(Y/N): \n");
        scanf("%c", &answer);
    }
```

```
    while (strchr("YyNn", answer) == NULL);
    return (answer == 'Y' || answer == 'y');
}

void printMessage(const char *format, ...)
{
    va_list ap;

    va_start(ap, format);
    fprintf(logFile, "> ");
    vfprintf(logFile, format, ap);
    va_end(ap);
    va_start(ap, format);
    vfprintf(stdout, format, ap);
    va_end(ap);
}

void printErr(const char *format, ...)
{
    va_list ap;

    va_start(ap, format);
    fprintf(logFile, "> ");
    vfprintf(logFile, format, ap);
    va_end(ap);
    va_start(ap, format);
    vfprintf(stderr, format, ap);
    va_end(ap);
}

void printBoard()
{
    int i;

    fprintf(stdout, "ci sono %d prove\n", getTotalBoxesNumber());
    fprintf(logFile, "> ci sono %d prove\n", getTotalBoxesNumber());
    for (i = 1; i <= getTotalBoxesNumber(); i++) {
        fprintf(stdout, "%3d : %s", i, getDescription(i));
        fprintf(logFile, "> %3d : %s", i, getDescription(i));
    }
}

void closeIoInterface()
{
    if (fclose(logFile) != 0) {
        perror("ERROR: in closing log file");
        exit(EXIT_FAILURE);
    }
}

void initIoInterface(char *logFileName)
{
    if ((logFile = fopen(logFileName, "w")) == NULL) {
        perror("ERROR: unable to open log file");
        exit(EXIT_FAILURE);
    }
}
```

3 Test

Test con parametri non validi

file di configurazione inesistente

```
$./game nonExistentConfFile logFile p1 p2 p3
ERROR: error while opening configuration file
$
```

file di configurazione malformato

```
$./game malformedConfFile1 log p1 p2 p3
ERROR: missing ':' in configuration file
$

$./game malformedConfFile2 log p1 p2 p3
ERROR:invalid box num("
") in configuration file
$
```

file di log non valido

```
$ ls -l invalidLogFile
---------- 1 feffo feffo 0 2008-10-04 15:59 invalidLogFile
$ ./game conf invalidLogFile p1 p2 p3
ERROR: unable to open log file: Permission denied
$
```

nessun giocatore

```
$ ./game conf log
ERROR: Wrong number of arguments.
USAGE: game CONFIGURATION_FILE_NAME
                          LOG_FILE_NAME [PLAYER_NAME; 1..*]
$
```

due giocatori con lo stesso nome

```
$ ./game conf log p1 p2 p1
ERROR: found two player with thesame name "p1"
$
```

Test di alcuni casi limite

tabellone con una sola casella

```
$ ./game oneBoard log p1 p2 p3
ci sono 1 prove
  1 : prova numero 1
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     0 |         attivo |
> |      2 |              p2 |     0 |         attivo |
> |      3 |              p3 |     0 |         attivo |
> -----------------------------------------------------
next step?(Y/N):
y
next step?(Y/N):

************************************
turno 1
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     0 |         attivo |
> |      2 |              p2 |     0 |         attivo |
> |      3 |              p3 |     0 |         attivo |
> -----------------------------------------------------
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     6 |      terminato |
> |      2 |              p3 |     5 |      terminato |
> |      3 |              p2 |     2 |      terminato |
> -----------------------------------------------------


CLASSIFICA:

> ____________________________________________________________________
> |  posizione |            name | prove superate | non superate |
> --------------------------------------------------------------------
> |          1 |              p1 |              0 |            0 |
> |          2 |              p3 |              0 |            0 |
> |          3 |              p2 |              0 |            0 |
> --------------------------------------------------------------------
feffo@feffo-laptop:~/lls/llsBase1.1.1.13/lls$
```

un solo giocotore

```
$ ./game conf1 log p1
```

```
ci sono 10 prove
  1 : descrizione della prova 1.1
  2 : descrizione della prova 1.2
  3 : descrizione della prova 1.3
  4 : descrizione della prova 1.4
  5 : descrizione della prova 1.5
  6 : descrizione della prova 1.6
  7 : descrizione della prova 1.7
  8 : descrizione della prova 1.8
  9 : descrizione della prova 1.9
 10 : descrizione della prova 1.10
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     0 |         attivo |
> -----------------------------------------------------
next step?(Y/N):
y

************************************
turno 1
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     0 |         attivo |
> -----------------------------------------------------
player p1: "descrizione della prova 1.4
"
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     4 |         attivo |
> -----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
1y

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 2
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     4 | prova superata |
> -----------------------------------------------------
player p1: "descrizione della prova 1.6
"
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     6 |         attivo |
> -----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
1n

next step?(Y/N):
```

```
next step?(Y/N):
y

************************************
turno 3
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     6 | fermo un turno |
> -----------------------------------------------------
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     6 |    da attivare |
> -----------------------------------------------------
next step?(Y/N):
next step?(Y/N):
y

************************************
turno 4
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |     6 |    da attivare |
> -----------------------------------------------------
player p1: "descrizione della prova 1.10
"
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |    10 |         attivo |
> -----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
1y

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 5
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |    10 | prova superata |
> -----------------------------------------------------
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |              p1 |    16 |      terminato |
> -----------------------------------------------------


CLASSIFICA:
```

```
> _____________________________________________________________
> |  posizione |            name | prove superate | non superate |
> -------------------------------------------------------------
> |          1 |              p1 |              2 |            1 |
> -------------------------------------------------------------
feffo@feffo-laptop:~/lls/llsBase1.1.1.13/lls$
```

Test di uso ordinario

```
$ ./game conf3 log player1 player2 player3 player4

ci sono 10 prove
  1 : descrizione della prova 3.1
  2 : descrizione della prova 3.2
  3 : descrizione della prova 3.3
  4 : descrizione della prova 3.4
  5 : descrizione della prova 3.5
  6 : descrizione della prova 3.6
  7 : descrizione della prova 3.7
  8 : descrizione della prova 3.8
  9 : descrizione della prova 3.9
 10 : descrizione della prova 3.10
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |         player1 |     0 |         attivo |
> |      2 |         player2 |     0 |         attivo |
> |      3 |         player3 |     0 |         attivo |
> |      4 |         player4 |     0 |         attivo |
> -----------------------------------------------------
next step?(Y/N):
y

************************************
turno 1
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |         player1 |     0 |         attivo |
> |      2 |         player2 |     0 |         attivo |
> |      3 |         player3 |     0 |         attivo |
> |      4 |         player4 |     0 |         attivo |
> -----------------------------------------------------
player player1: "descrizione della prova 3.4
"
player player2: "descrizione della prova 3.1
"
player player3: "descrizione della prova 3.2
"
player player4: "descrizione della prova 3.3
"
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |         player1 |     4 |         attivo |
> |      2 |         player2 |     1 |         attivo |
> |      3 |         player3 |     2 |         attivo |
> |      4 |         player4 |     3 |         attivo |
> -----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
4n

numero del giocatore seguito dall'esito della prova Y/N:
3y

numero del giocatore seguito dall'esito della prova Y/N:
2n
```

```
numero del giocatore seguito dall'esito della prova Y/N:
1n

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 2
************************************
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |     4 | fermo un turno |
> |      2 |         player2 |     1 | fermo un turno |
> |      3 |         player3 |     2 | prova superata |
> |      4 |         player4 |     3 | fermo un turno |
> ----------------------------------------------------
player player3: "descrizione della prova 3.8
"
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |     4 |    da attivare |
> |      2 |         player2 |     1 |    da attivare |
> |      3 |         player3 |     8 |         attivo |
> |      4 |         player4 |     3 |    da attivare |
> ----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
3n

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 3
************************************
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |     4 |    da attivare |
> |      2 |         player2 |     1 |    da attivare |
> |      3 |         player3 |     8 | fermo un turno |
> |      4 |         player4 |     3 |    da attivare |
> ----------------------------------------------------
player player1: "descrizione della prova 3.10
"
player player2: "descrizione della prova 3.2
"
player player4: "descrizione della prova 3.5
"
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |    10 |         attivo |
> |      2 |         player2 |     2 |         attivo |
> |      3 |         player3 |     8 |    da attivare |
> |      4 |         player4 |     5 |         attivo |
> ----------------------------------------------------
```

```
numero del giocatore seguito dall'esito della prova Y/N:
1y

numero del giocatore seguito dall'esito della prova Y/N:
3n

WARNING: player number 3 not valid because player:
      -won
      -is out of turn
      -already passed the test
numero del giocatore seguito dall'esito della prova Y/N:
2y

numero del giocatore seguito dall'esito della prova Y/N:
4y

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 4
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |         player1 |    10 | prova superata |
> |      2 |         player2 |     2 | prova superata |
> |      3 |         player3 |     8 |    da attivare |
> |      4 |         player4 |     5 | prova superata |
> -----------------------------------------------------
player player2: "descrizione della prova 3.3
"
player player4: "descrizione della prova 3.8
"
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
> |      2 |         player2 |     3 |         attivo |
> |      3 |         player3 |    14 |      terminato |
> |      4 |         player4 |     8 |         attivo |
> -----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
2y

numero del giocatore seguito dall'esito della prova Y/N:
4y

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 5
************************************
> _____________________________________________________
> | number |            name |   box |          state |
> -----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
```

```
> |      2 |         player2 |     3 | prova superata |
> |      3 |         player3 |    14 |      terminato |
> |      4 |         player4 |     8 | prova superata |
> ------------------------------------------------------
player player2: "descrizione della prova 3.6
"
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
> |      2 |         player2 |     6 |         attivo |
> |      3 |         player3 |    14 |      terminato |
> |      4 |         player4 |    13 |      terminato |
> ----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
2y

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 6
************************************
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
> |      2 |         player2 |     6 | prova superata |
> |      3 |         player3 |    14 |      terminato |
> |      4 |         player4 |    13 |      terminato |
> ----------------------------------------------------
player player2: "descrizione della prova 3.8
"
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
> |      2 |         player2 |     8 |         attivo |
> |      3 |         player3 |    14 |      terminato |
> |      4 |         player4 |    13 |      terminato |
> ----------------------------------------------------
numero del giocatore seguito dall'esito della prova Y/N:
2y

next step?(Y/N):
next step?(Y/N):
y

************************************
turno 7
************************************
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
> |      2 |         player2 |     8 | prova superata |
> |      3 |         player3 |    14 |      terminato |
> |      4 |         player4 |    13 |      terminato |
> ----------------------------------------------------
```

```
> ____________________________________________________
> | number |            name |   box |          state |
> ----------------------------------------------------
> |      1 |         player1 |    15 |      terminato |
> |      2 |         player3 |    14 |      terminato |
> |      3 |         player4 |    13 |      terminato |
> |      4 |         player2 |    11 |      terminato |
> ----------------------------------------------------


CLASSIFICA:

> ____________________________________________________________________
> |  posizione |            name | prove superate | non superate |
> --------------------------------------------------------------------
> |          1 |         player1 |              1 |            1 |
> |          2 |         player3 |              1 |            1 |
> |          3 |         player4 |              2 |            1 |
> |          4 |         player2 |              4 |            1 |
> --------------------------------------------------------------------
feffo@feffo-laptop:~/lls/llsBase1.1.1.13/lls$
```

4 Contenuto dei files di test

malformedConfFile1

```
1: prova 1
2: prova 2
3  prova 3
```

---

malformedConfFile2

```
1: prova 1
2: prova 2

3:  prova 3
```

---

oneBoard

```
1: prova numero 1
```

---

conf1

```
01: descrizione della prova 1.1
02: descrizione della prova 1.2
03: descrizione della prova 1.3
04: descrizione della prova 1.4
05: descrizione della prova 1.5
06: descrizione della prova 1.6
07: descrizione della prova 1.7
08: descrizione della prova 1.8
09: descrizione della prova 1.9
10: descrizione della prova 1.10
```

---

conf3

```
01: descrizione della prova 3.1
02: descrizione della prova 3.2
03: descrizione della prova 3.3
04: descrizione della prova 3.4
05: descrizione della prova 3.5
06: descrizione della prova 3.6
07: descrizione della prova 3.7
08: descrizione della prova 3.8
09: descrizione della prova 3.9
10: descrizione della prova 3.10
```