Informatica – I Facoltà di Ingegneria
Prova scritta del 20/01/2014 – TURNO B

Il Gomoku è un gioco tradizionale giapponese di allineamento. Due giocatori posano alternativamente le loro pietre su una scacchiera 19x19 e vince il primo che riesce a mettere 5 pietre del suo colore in fila (in orizzontale o in verticale).

Si chiede di realizzare un programma che permetta di riprendere una partita a Gomoku precedentemente salvata e portarla a termine. Il programma, all'avvio, carica lo stato della partita da un file denominato "saved.dat".

Successivamente il programma deve permettere ai due giocatori di continuare la partita, chiedendogli (alternativamente, prima al nero e poi al bianco) le coordinate della casella su cui posare la propria pietra, e controllando se la mossa porta alla vittoria. In tal caso il programma termina comunicando "Partita terminata: vittoria del nero!" (o "del bianco"). L'utente inserisce le coordinate (due numeri interi) separandole con uno spazio.

Per ogni mossa, il programma deve inoltre verificare che la mossa sia corretta, che cioè le coordinate siano comprese fra 1 e 19 (inclusi) e che la casella sia libera. Se la mossa non è corretta il programma stampa a video "Mossa non valida." e chiede nuovamente allo stesso giocatore di inserire le coordinate della mossa.

Il file che contiene la situazione del gioco contiene una riga di caratteri per ogni riga della scacchiera; qui il carattere punto '.' rappresenta una casella ancora libera, il carattere 'B' una pietra del bianco, il carattere 'N' una pietra del nero. Dopo aver caricato la situazione da file tocca sempre giocare al nero. Il formato del file è corretto.

File saved.dat (solo le prime 5 righe)

```
..................
..................
...B....B.........
...B..............
.....N..NN........
```

Esecuzione del programma

```
C:\> GOMOKU.EXE
Inserire la mossa del nero (R C): 5 7
Inserire la mossa del bianco (R C): 3 4
Mossa non valida.
Inserire la mossa del bianco (R C): 1 1
Inserire la mossa del nero (R C): 5 8
Partita terminata: vittoria del nero!
```

#include <stdio.h>

FILE *fopen(char *filename, char * mode) – Apertura di un file (mode: “r” lettura – “w” scrittura – “a” append)
FILE *freopen(char *filename, char * mode, FILE *file_pointer) - Riassegna un file puntatore ad un file diverso.
int fclose(FILE *file_pointer) - Chiude un file
int feof(FILE *file_pointer) - Controlla se e' stato incontrato un end-of-file in un file.
int fflush(FILE *file_pointer) - Svuota il buffer di un file.
int getchar(void) - Legge un carattere da "stdin" (tastiera)
int fgetc(FILE *file_pointer) - Prende un carattere da un file
char *gets(char *buffer) - Legge una riga da "stdin" (tastiera)
char *fgets(char *string, int maxchar, FILE *file_pointer) - Legge una riga da un file.
int printf(char *format _string, …) - Scrive output formattato su "stdout" (schermo)
int fprintf(FILE *file_pointer, char *format_string, …) - Scrive output formattato in un file.
int sprintf(char *string, char *format_string, …) - Scrive output formattato su una stringa
int fputc(int c, FILE *file_pointer) - Scrive un carattere in un file
int putchar(int c) - Scrive un carattere su "stdout" (schermo)
int puts(char *string) - Scrive una stringa su "stdout" (schermo)
int fputs(char *string, FILE *file_pointer) - Scrive una stringa in un file.
int scanf(char *format_string, args) - Legge input formattato da "stdin" (tastiera)
int fscanf(FILE *file_pointer, char *format string, args) - Legge input formattato da file
int sscanf(char *buffer, char *format_string, args) - Legge input formattato da una stringa
EOF – end of file (costante a valore negativo)
NULL - puntatore nullo (valore 0)

#include <stdlib.h>

double atof(char *string) - Converte una stringa in un valore in floating point.
int atoi(char *string) - Converte una stringa in un valore integer.
int atol(char *string) - Converte una stringa in un valore long integer.
void exit(int val) – Termina il programma, restituendo il valore ‘val’.
EXIT_FAILURE - costante per segnalare terminazione senza successo del programma con exit(); valore diverso da zero
EXIT_SUCCESS - segnala terminazione con successo del programma con exit(); vale 0

#include <string.h>

char *stpcpy (char *dest, char *src) - Copia una stringa in un'altra. Restituisce dest
char *strncpy(char *s1, char *s2, size_t n) - Copia i primi "n" caratteri di s2 in s1. Restituisce s1
int strcmp(char *s1, char *s2) - Confronta s1 e s2 per determinare l'ordine alfabetico (<0, s1 prima di s2, 0 uguali, >0 s1 dopo s2)
int strncmp(char *s1, char *s2, size_t n) - Confronta i primi "n" caratteri di due stringhe.
char *strcpy(char *s1, char *s2) - Copia s2 in s1. Restituisce s1
int strlen(char *string) - Determina la lunghezza di una stringa.
char *strcat(char *s1, char *s2, size_t n) - Aggiunge s2 a s1. Ritorna s1
char *strncat(char *s1, char *s2, size_t n) - Aggiunge "n" caratteri di s2 a s1. Ritorna s1
char *strchr(char *string, int c) - Cerca la prima occorrenza del carattere ‘c’ in string; restituisce un puntatore alla prima occorrenza di c in s, NULL se non presente
char *strrchr(char *string, int c) - Cerca l'ultima occorrenza del carattere ‘c’ in string
char* strstr(char* s, char* t) - Restituisce un puntatore alla prima occorrenza di t all'interno di s. Restituisce NULL se t non è presente in s.
char* strtok(char* s, const char* t) - scompone s in token, i caratteri che delimitano i token sono contenuti in t. Restituisce il puntatore al token (NULL se non ne trova nessuno). Alla prima chiamata in s va inserita la stringa da scomporre e in t i caratteri che delimitano i vari token. Per operare sulla stessa stringa, alle successive chiamate al posto di s si deve passare NULL

#include <ctype.h>

int isalnum(int c) - Vero se ‘c’ e' alfanumerico.
int isalpha(int c) - Vero se ‘c’ e' una lettera dell'alfabeto.
int iscntrl(int c) - Vero se ‘c’ e' un carattere di controllo.
int isdigit(int c) - Vero se ‘c’ e' un numero decimale.
int islower(int c) - Vero se ‘c’ e' una lettera minuscola.
int isprint(int c) - Vero se ‘c’ e' un carattere stampabile.
int ispunct (int c) - Vero se ‘c’ e' un carattere di punteggiatura.
int isspace(int c) - Vero se ‘c’ e' un carattere spazio.
int isupper(int c) - Vero se ‘c’ e' una lettera maiuscola.
tolower(int c) - Converte ‘c’ in minuscolo.
int toupper(int c) - Converte ‘c’ in maiuscolo.

#include <math.h>

int abs (int n) – valore assoluto intero
long labs(long n) – valore assoluto long
double fabs (double x) – valore assoluto di x
double acos(double x) - arcocoseno
double asin(double x) - arcoseno
double atan(double x) - arcotangente
double atan2(double y, double x) – arcotangente di y/x.
double ceil(double x) – intero superiore a x
double floor(double x) – intero inferiore a x.
double cos(double x) – x in radianti
double sin(double x) – x in radianti
double tan(double x) – x in radianti
double cosh(double x) – coseno iperbolico
double sinh(double x) – seno iperbolico
double tanh(double x) – tangente iperbolica
double exp(double x) - e^x
double log(double x) - log(x).
double log10 (double x) – logaritmo base 10
double pow (double x, double y) - x^y
int rand (void) – intero casuale tra 0 e RND_MAX.
int random(int max_num) – valore casuale tra 0 e max_num.
void srand(unsigned seed) – inizializza la sequenza di valori casuali
double sqrt(double x) – radice quadrata

#include <limits.h>

INT_MAX - Indica il più grande valore che è possibile rappresentare con un int.
INT_MIN - Indica il più piccolo valore che è possibile rappresentare con un int.
LONG_MAX - Indica il più grande valore che è possibile rappresentare con un long.
LONG_MIN - Indica il più piccolo valore che è possibile rappresentare con un long.

#include <float.h>

FLT_MAX, DBL_MAX - Indica il più grande valore che è possibile rappresentare con un float (o double)
FLT_MIN, DBL_MIN - Indica il più piccolo valore che è possibile rappresentare con un float (o double)

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

#define SIZE    19
#define L        5

#define FREE    '.'
#define WHITE   'B'
#define BLACK   'N'

#define STR_LEN   6
#define WHITE_STR "bianco"
#define BLACK_STR "nero"

#define VICTORY_STR "Partita terminata: vittoria del %s!\n"
#define NOT_VALID_MOVE_STR "Mossa non valida: ...\n"

#define TRUE    1
#define FALSE   0

char players[2][STR_LEN+1] = { BLACK_STR, WHITE_STR };
char symbols[2] = { BLACK, WHITE };

char board[SIZE][SIZE+1];       // 19 characters + /n + /0

void print_board( char b[][SIZE+1] ) {
        int i, j;
        printf("%02d ", 0 );
        for(j=0; j<SIZE; j++) {
                printf("%02d ", j + 1);
        }
        printf("\n");
        for(i=0; i<SIZE; i++) {
                printf("%02d ", i + 1);
                for(j=0; j<SIZE; j++) {
                        printf(" %c ",b[i][j]);
                }
                printf("\n");
        }
}

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

        FILE *fp;

        int i;
        int r, c;       // row, column

        int cur_player = 0;
        int victory;
        int valid;
        int cnt;

        if( (fp = fopen("saved.dat","r")) == NULL ) {
                printf("Input file not found.\n");
                exit(EXIT_FAILURE);
        }

        for(i=0; i<SIZE; i++) {
                fscanf(fp, "%s", board[i]);
        }


        while(1) {

                print_board( board );

                // read move
                do {
                        valid = TRUE;
                        printf("Inserire la mossa del %s (R/C): ", players[cur_player]);
                        scanf("%d %d", &r, &c);

                        // check the move
```

```
                        if( r<1 || r>SIZE || c<1 || c>SIZE || board[r-1][c-1] != FREE) {
                                printf(NOT_VALID_MOVE_STR);
                                valid = FALSE;
                        }
                }while(!valid);

                // update the board
                board[r-1][c-1] = symbols[cur_player];

                // check for victory in each direction
                victory = FALSE;

                // check horizontally
                cnt = 0;
                for(i=0;i<SIZE && !victory;i++) {
                        // if, in a row, we find L consecutive (horizontal) cells
                        // with the same symbol of the current player ('N' or 'B')
                        if( board[r-1][i] == board[r-1][c-1] ) {
                                cnt++;
                                if(cnt == L) {
                                        victory = TRUE;
                                        printf("Partita terminata: vittoria del %s!\n", players
[cur_player]);
                                }
                        } else {
                                // if a cell has a different symbol, reset the counter to 0
                                cnt = 0;
                        }
                }

                // check vertically
                cnt = 0;
                for(i=0;i<SIZE && !victory;i++) {
                        if( board[i][c-1] == board[r-1][c-1] ) {
                                cnt++;
                                if(cnt == L) {
                                        victory = TRUE;
                                        printf("Partita terminata: vittoria del %s!\n", players
[cur_player]);
                                }
                        } else {
                                cnt = 0;
                        }
                }


                if(!victory) {
                        // next player
                        // 0 is black, 1 is white
                        cur_player = (cur_player + 1 ) % 2;
                } else {
                        break;
                }

        }

        print_board( board );


        fclose(fp);

        return EXIT_SUCCESS;
}
```