# Othello

## Progetto d'esame del corso di "Programmazione e Laboratorio"

## 1 Introduzione

Scopo del progetto è realizzare un programma che impersoni un giocatore (di una variante semplificata) del gioco Othello. Il programma legge le mosse dell'avversario dallo *standard input* e stampa le proprie mosse sullo *standard output* attenendosi ad un preciso formato. Lo studente ha a disposizione un programma in grado di arbitrare una partita tra due giocatori che può essere facoltativamente utilizzato per sperimentare la correttezza e la qualità della strategia di gioco implementata. Una volta ottenuto un programma che compila correttamente e funziona in modo soddisfacente, lo studente consegna al docente, in modo opportuno, i file sorgenti necessari alla sua compilazione.

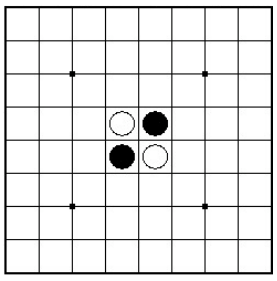
**Chi intende sostenere l'esame nell'appello del 12 aprile 2006 deve effettuare la consegna entro e non oltre il 5 aprile 2006.**

Nelle sezioni seguenti verranno specificate: le regole del gioco, il formato dell'input/output, le modalità per utilizzare il programma arbitro e per effettuare la consegna.

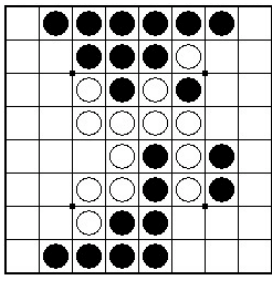
## 2 Regole del gioco

Il gioco denominato Othello è stato inventato, come variante dell'antico gioco inglese Reversi, dal nipponico Goro Hasegawa, nel 1971. La versione considerata in questo progetto risulta semplificata rispetto a quella originale; il giocatore implementato *deve attenersi strettamente alle regole specificate di seguito* (e non a quelle di nessuna altra versione, per quanto ufficiale, o filologicamente esatta).

Il gioco ha due giocatori che dispongono una serie di pedine bicolori, con un lato bianco ed uno nero, su una scacchiera $8 \times 8$. Le righe e colonne della scacchiera sono numerate da 0 a 7 da sinistra a destra e dall'alto verso il basso. Un giocatore è il nero, l'avversario il bianco. La disposizione iniziale delle pedine sulla scacchiera è quella della Figura (a). Inizia a giocare il nero. Al suo turno ogni giocatore compie la sua mossa, ossia appoggia una pedina, con la



(a) Posizione iniziale.



(b) Posizione intermedia.

faccia del proprio colore rivolta verso l'alto, su una casella ancora vuota in modo da imprigionare, tra la pedina che sta giocando e quelle del proprio colore già presenti sulla scacchiera, una o più pedine del colore dell'avversario. A questo punto le pedine imprigionate vengono rovesciate diventando così del colore di chi ha eseguito la mossa. È possibile imprigionare le pedine in orizzontale e in verticale (ma *non in diagonale*) e, a ogni mossa, si possono imprigionare (e quindi girare) pedine in una o più direzioni contemporaneamente.

Ad esempio, se ad un certo punto la scacchiera si presenta come in Figura (b) il bianco può mettere una pedina nella casella di riga 1 e colonna 1 imprigionando (e quindi girando) tre pedine nere in orizzontale (alla riga 1 e colonna 2, 3 e 4); similmente, il nero può mettere una pedina nella casella di riga 4 e colonna 2 imprigionando (e quindi girando) cinque pedine bianche, di cui quattro in verticale (alla colonna 2 e riga 2, 3, 5 e 6) ed una in orizzontale (alla riga 4 e colonna 3).

Il gioco termina quando il giocatore di turno non può disporre la sua pedina o perché la scacchiera è piena, o perché non ha modo di imprigionare alcuna pedina dell'avversario. Vince il giocatore che, al termine del gioco, ha il maggior numero di pedine del proprio colore sulla scacchiera (è patta se entrambi hanno lo stesso numero di pedine); più precisamente, ogni giocatore riporta come punteggio al termine della partita il numero di pedine del suo colore presenti sulla scacchiera al termine del gioco.

# 3 Formato dell'input/output

All'inizio di ogni partita, il programma legge da *standard input* un numero intero che può essere 1, o 0, e che indica se il giocatore è rispettivamente il nero, o il bianco (ossia se è il primo a giocare, o meno). Se il numero letto indica che il giocatore è il nero, il programma scrive su *standard output* la riga e colonna della sua prima mossa; quindi inizia un ciclo in cui legge da *standard input* la riga e colonna della mossa dell'avversario e risponde scrivendo su *standard output* la riga e colonna della sua mossa di risposta, fino a quando viene indicato il termine della partita. A questo punto, il programma ricomincia una nuova partita, seguendo da capo il processo descritto in questo paragrafo (quindi non termina mai).

Il programma deve predisporsi all'input/output chiamando (nel corpo della funzione `main`, subito dopo l'eventuale dichiarazione delle variabili locali e *prima di qualunque altra istruzione*) la funzione `setbuf` (definita nel file di intestazione `stdio.h`) con i seguenti parametri: `setbuf( stdin, NULL )` e `setbuf( stdout, NULL )`. Il primo intero (che determina il colore del giocatore) deve essere quindi letto utilizzando la funzione `scanf` con stringa di formato `"%d"` mentre le mosse devono essere lette utilizzando la funzione `scanf` con stringa di formato `"%d %d"`, dove il primo intero letto corrisponde al numero di riga della mossa e il secondo al numero di colonna. Analogamente, le mosse devono essere scritte utilizzando la funzione `printf` con stringa di formato `"%d %d\n"`, dove il primo intero scritto corrisponde al numero di riga della mossa e il secondo al numero di colonna. I numeri di riga e colonna sono compresi tra 0 e 7 (estremi inclusi), oppure sono entrambi uguali a $-1$ ad indicare che la partita è terminata.

Si riporta come suggerimento un possibile "scheletro" di programma giocatore:

```
int p, r, c;
...
setbuf( stdin, NULL );
setbuf( stdout, NULL );
...
for (;;) {
        ...
        /* lettura per determinare il colore */
        if ( scanf( "%d", &p ) != 1 ) return -1;
        if ( p == 1 ) {
                ...
                /* scrittura prima mossa */
                printf( "%d %d\n", r, c );
                ...
        } else {
                ...
        }
        ...
        for (;;) {
                ...
                /* lettura mossa avversario */
                if ( scanf( "%d %d", &r, &c ) != 2 ) return -1;
                if ( r == -1 && c == -1 ) break;
                ...
                /* scrittura mossa risposta */
                printf( "%d %d\n", r, c );
                ...
        }
        ...
}
```

Si osserva che nello scheletro compaiono esplicitamente tutte le funzioni di input/output necessarie e sufficienti per svolgere il progetto. Il programma scritto dallo studente può differire dallo scheletro suggerito (che deve essere ovviamente integrato di tutto quello che serve per implementare la strategia di gioco!), ma in nessun caso deve produrre altro output, o attendersi altro input, rispetto a quello prodotto dalle due chiamate a `scanf` e `printf` dello scheletro precedente.

Il programma può assumere che tutte le mosse che legge dallo standard input sono valide e deve tassativamente garantire che tutte le mosse che scrive sullo standard output siano valide (una mossa è valida se fa riferimento a una casella vuota e intrappola almeno una pedina avversaria). Il programma può inoltre assumere che se non c'è nessuna mossa di risposta valida (perché la scacchiera è piena, o perché non è possibile imprigionare nessuna pedina avversaria), l'ultima mossa letta è quella di termine della partita (ossia quella in cui il numero di riga e colonna è pari a $-1$).

A titolo di esempio, si riportano due possibili esecuzioni parziali del programma dove le parti in grassetto corrispondono a quanto il programma legge da *standard input*, mentre le altre (non in grassetto) a quanto scrive su *standard output* (i puntini "$\cdots$" indicano le mosse non riportate):

| |
|---|
| **1** |
| 2 3 |
| **4 2** |
| 5 3 |
| $\cdots$ |
| **-1 -1** |

| |
|---|
| **0** |
| **2 3** |
| 4 2 |
| **5 3** |
| $\cdots$ |
| **-1 -1** |

Nell'esempio di sinistra, il programma gioca come nero, quindi inizia stampando la sua mossa, viceversa, nell'esempio di destra, gioca come bianco, quindi inizia leggendo la mossa dell'avversario.

Si ribadisce nuovamente che il formato dell'input/output deve attenersi strettamente a quanto specificato in questa sezione, che non deve essere letto, o scritto, null'altro che numeri interi, separati da spazi e "a capo" esattamente come specificato e come mostrato nell'esempio.

**In mancanza di completa e stretta aderenza alle indicazioni precedenti il progetto non sarà ritenuto valido.**

# 4 Uso del programma arbitro

Il docente mette a disposizione, tramite le risorse di calcolo del laboratorio SILab, un programma denominato `giocatore` e uno denominato `arbitro` che implementano rispettivamente un semplice giocatore e un arbitro in grado di gestire una sfida tra due giocatori; i programmi in forma eseguibile si trovano entrambi nella directory `/users/ms000123/`. Il loro uso è facoltativo e va inteso come supporto allo sviluppo del progetto.

Il programma `giocatore` corrisponde ad una semplice implementazione del progetto d'esame e il suo input/output si attiene a quanto specificato nella sezione precedente. Può essere utilizzato direttamente ad esempio per rendersi conto del formato corretto dell'input/output, come mostrato nell'esempio della sezione precedente.

Il programma `arbitro` può essere utilizzato per confrontare le prestazioni di due programmi di gioco. Va invocato con due parametri corrispondenti ciascuno ad un programma giocatore, nel qual caso verrà svolta una sola partita, alternativamente il programma va invocato con un terzo parametro pari al numero di partite da giocare. L'arbitro produce su *standard output* l'elenco (in formato HTML) delle configurazioni finali delle scacchiere prodotte nelle varie partite o, se invocato con l'opzione `-v`, dell'intera sequenza di scacchiere aggiornate di mossa in mossa. In ogni caso, al termine dell'esecuzione, sullo *standard error* viene riportato il punteggio finale dei due giocatori.

Se, ad esempio, uno studente avesse compilato il suo programma con nome `campione` nella sua home, alcune invocazioni potrebbero essere

```
$ cd
$ /users/ms000123/arbitro -v ./campione /users/ms000123/giocatore > dettaglio.html
$ /users/ms000123/arbitro ./campione /users/ms000123/giocatore 10 > dieci.html
$ /users/ms000123/arbitro -v ./campione ./campione 2 > sfida.html
```

La prima, genera il file `dettaglio.html` che (aperto con un browser) mostra lo svolgimento mossa per mossa di una partita disputata dal programma dello studente contro quello del docente, mentre la seconda genera il file `dieci.html` che mostra dieci configurazioni finali della scacchiera di altrettante partite disputate dai due programmi; in fine, l'ultima invocazione genera il file `sfida.html` che mostra le configurazioni finali della scacchiera di due partite disputate dal programma dello studente contro se stesso.

# 5 Modalità di consegna

La consegna del progetto avviene tramite le risorse di calcolo del laboratorio SILab. Lo studente che avesse sviluppato il progetto utilizzando risorse diverse (ad esempio il proprio computer di casa), dovrà trasferirne i file, prima della consegna, nella sua home del SILab ed accertarsi che il suo programma compili e funzioni correttamente in tale ambiente.

**Programmi consegnati diversamente da come specificato in questa sezione, o che non compilano ed eseguono correttamente nell'ambiente di calcolo del SILab non saranno ritenuti validi.**

Più precisamente, deve essere possibile compilare il programma invocando il comando `gcc -lm -Wall *.c` in una directory che contenga tutti e soli i file sorgenti relativi al progetto. In particolare, se il programma si basa su più di un file sorgente, tra quelli con estensione `.c` deve esisterne uno e uno solo che contenga una funzione di nome `main`. Una volta compilato il programma, la sua esecuzione tramite il programma `arbitro` (in sfida con un altro programma corretto) non deve produrre errori.

Lo studente deve consegnare il progetto invocando il comando `/users/ms000123/consegna` messo a disposizione dal docente. Il comando va invocato passando come parametri i nomi di tutti e soli i file sorgenti necessari alla compilazione del programma.

Assumendo ad esempio che tali file siano contenuti nella directory corrente e abbiano nome `gioca.c`, `lib.c` e `lib.h`, la corretta invocazione è

```
$ /users/ms000123/consegna gioca.c lib.c lib.h
```

l'output del comando indica se la consegna è andata a buon fine, oppure se non è avvenuta (a causa, ad esempio, di un errore di compilazione).

Il comando `/users/ms000123/verifica` messo a disposizione dal docente può essere invocato, senza argomenti, per verificare l'esito della consegna.

Lo studente è libero di effettuare più volte la consegna, **in caso di consegne multiple, verrà considerata valida solo l'ultima consegna in ordine temporale**. Poiché il processo di consegna potrebbe non andare a buon fine al primo tentativo (a causa di errori vari), si consiglia vivamente di provare a consegnare una versione preliminare del progetto appena possibile, per non trovarsi a dover risolvere problemi inaspettati in prossimità della scadenza.

Si ricorda che chi intende sostenere l'esame nell'appello del 12 aprile 2006 deve consegnare il progetto entro e non oltre il **5 aprile 2006**. Fa fede la data riportata dal comando `/users/ms000123/verifica`.

Una volta consegnata la versione definitiva del progetto lo studente deve *stampare una copia di tutti i file sorgenti che porterà con se all'esame orale*.