Lezione 4

La lezione è centrata su un esercizio per impratichirsi sui seguenti argomenti:

- gestione di vettori statici multidimensionali

Chi notasse eventuali incoerenze o errori, oppure avesse dubbi sul contenuto di queste pagine e dei codici, è pregato di segnalarmeli per contribuire a migliorare la qualità dei materiali del corso.

Problema

Othello è un gioco da tavolo. Si gioca in due, su una scacchiera quadrata di 8 caselle per lato, le cui righe sono associate alle cifre da 1 a 8, mentre le colonne sono associate alle lettere da *a* ad *h*. I giocatori usano pedine con un lato nero e l'altro bianco: il giocatore Nero le posa sulla scacchiera col lato nero volto verso l'alto, il giocatore Bianco le posa col lato bianco verso l'alto. Si parte da una configurazione in cui le caselle *d4* ed *e5* sono occupate da pedine bianche, mentre le caselle *d5* ed *e4* sono occupate da pedine nere. Muove per primo il giocatore Nero. Ogni giocatore deve catturare almeno una pedina di colore opposto: per farlo, la nuova pedina e una pedina vecchia dello stesso colore devono racchiudere fra loro una sequenza ininterrotta (orizzontale, verticale o diagonale) di una o più pedine del colore opposto. Al termine della mossa, tutte le pedine della sequenza risultano catturate e vengono girate in modo da assumere il colore del giocatore che ha appena mosso. Con una sola mossa si possono catturare più sequenze di pedine contemporaneamente, se vi sono diverse pedine vecchie che formano sequenze utili con la nuova. Se un giocatore non ha mosse lecite, passa il turno. Se entrambi non hanno mosse lecite (per esempio, perché la scacchiera è interamente piena), il gioco si conclude con il conteggio delle pedine: vince il giocatore che ha più pedine del proprio colore. La figura 1 indica la configurazione iniziale della scacchiera e le mosse lecite per il giocatore Nero; la figura 2 indica la configurazione raggiunta dopo l'esecuzione della mossa *d3* e le mosse lecite per il giocatore Bianco.

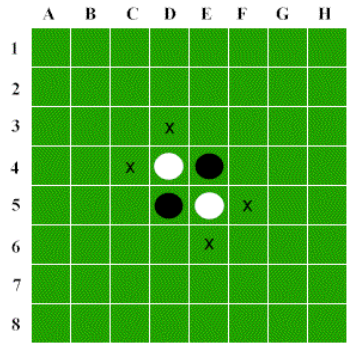

Figura 1: Configurazione iniziale: le crocette indicano le mosse lecite per il giocatore Nero, cioè *c4*, *d3*, *e6*, *f5*

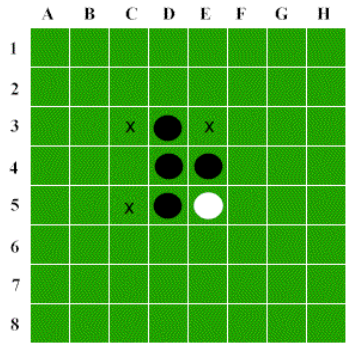

Figura 2: Configurazione della scacchiera dopo che il giocatore Nero ha eseguito la mossa *d3*: le crocette indicano le mosse lecite per il giocatore Bianco, cioè *c3*, *c5*, *e3*

Si vuole realizzare un algoritmo che valuti la correttezza di una sequenza di mosse, che le esegua se sono corrette e che al termine indichi la situazione del gioco.

Formato di ingresso e di uscita La sequenza di mosse da valutare ed eseguire è riportata in un file di testo, che codifica le mosse secondo il seguente formato. Una mossa regolare corrisponde a una stringa di 2 caratteri: una lettera compresa fra `a` e `h`, che indica la colonna, immediatamente seguita da una cifra compresa fra `1` e `8`, che indica la riga. Due trattini consecutivi indicano che un giocatore passa il turno perché non ha mosse lecite. Le mosse sono separate da spazi, tabulazioni o a capi. Ad esempio:

```
c4 e3 f2 b4 d3 c5
a4 e2 f1 e1 d1 g1
h1 d2 e6 c3 d6 f3
b2 f4 g4 g3 h3 f5
f6 g2 c1 c2 h2 --
```

La stampa a video dei risultati usa invece il seguente formato. Se la sequenza di mosse non è corretta, si deve stampare a video la mossa non lecita e il giocatore a cui è stata attribuita. Altrimenti, si stampa la configurazione finale della scacchiera: la prima riga contiene i caratteri che corrispondono alle colonne; ciascuna riga seguente comincia con la cifra associata, seguita da 8 caratteri che descrivono lo stato di ogni casella. Se la casella contiene una pedina nera, si stampa il carattere `N`; se ne contiene una bianca, il carattere `B`; se è vuota, il carattere `.` (punto). Alla configurazione segue una riga vuota, una riga che fornisce il numero corrente di pedine nere seguito da `pedine nere` e una riga che fornisce il numero corrente di pedine bianche seguito da `pedine bianche`. Il tutto si conclude con l'indicazione dell'esito della partita, che può essere `Vince il Nero`, `Vince il Bianco` o `Pareggio`. La partita precedente è corretta e si conclude con la vittoria del Nero per 33 pedine a 0. Il programma dovrà quindi stampare a video:

```
 abcdefgh
1..NNNNNN
2.NNNNNNN
3..NNNNNN
4NNNNNNN.
5..NNNN..
6...NNN..
7........
8........

33 pedine nere
0 pedine bianche
Vince il Nero
```

Traccia della risoluzione

Prima di eseguire l'esercizio, conviene scorrere i lucidi della lezione 4 sulle strutture dati astratte, sulla definizione astratta e sulle implementazioni in C dei vettori e dei record.

Il punto di partenza dell'intero esercizio è il file `othello0.c`, che già contiene:

1. un `main` con la variabile `filemosse` che conserva il nome del file delle mosse;
2. la funzione `InterpretaLineaComando` che riceve il nome dalla linea di comando (`argv[1]`) e lo copia nella variabile;

3. le direttive per simulare il tipo `boolean`;

4. l'istruzione `typedef char Scacchiera[BOARD_SIZE][BOARD_SIZE];` e le costanti simboliche `BOARD_SIZE`, `BIANCO`, `NERO` e `LIBERO` per gestire la scacchiera;

5. la costante simbolica `MOVE_LENGTH` per gestire le mosse.

Tratteremo infatti la scacchiera come una struttura dati astratta, dotata di un suo nome, di funzioni con le quali operare su di essa (per esempio, per leggerla, stamparla a video, modificarla, valutare il vincitore, ecc...) e di costanti simboliche per poterne modificare le caratteristiche senza dover riscrivere l'intero codice (per esempio, per cambiare i simboli usati per i due giocatori, o le dimensioni della scacchiera). Contrariamente a quanto suggerito sui lucidi, indicizziamo righe e colonne da 0 a 7, anziché da 1 a 8, perché le mosse vengono rappresentate da coppie di caratteri e risulta quindi comodo sfruttare l'"aritmetica dei puntatori" per convertire la lettera che indica una riga nell'indice corrispondente (sottraendole la `'a'`) e la cifra che rappresenta ogni colonna nell'indice corrispondente (sottraendole la cifra `'1'`), e viceversa.

Prima fase (`othello1.c`) Si avvia la realizzazione *top-down* dell'algoritmo scrivendo opportune funzioni e aggiungendo al codice la loro chiamata nel `main`, la dichiarazione fra i prototipi prima del `main`, la definizione dopo il `main`:

1. inizializzazione della scacchiera

2. esecuzione (e controllo) delle mosse indicate nel file

3. stampa della scacchiera finale

4. stampa dell'esito del gioco

Inoltre, si aggiunge al `main` la dichiarazione della scacchiera `S`, in modo da definire completamente i passaggi di dati e risultati da una procedura all'altro.

Seconda fase (`othello2.c`) Ora, vincendo la tentazione di scrivere il codice dall'inizio alla fine, procederemo invece "a cipolla", cioè dalle procedure più esterne a quelle più interne. In altre parole, cominceremo realizzando:

- l'inizializzazione della scacchiera, dunque l'operazione di sostituzione (scrittura): usare le costanti simboliche consente di scrivere codice valido per scacchiere di dimensione varia e rappresentate con simboli vari;

- stampa della scacchiera: richiede la banale operazione di proiezione (lettura), ma si può complicare la stampa aggiungendo una cornice con le lettere associate alle colonne e le cifre associate alle righe: per determinare la lettera e la cifra associata si può usare l'*aritmetica dei puntatori*;

- stampa dell'esito del gioco: richiede un conteggio delle caselle bianche e nere, e un confronto del risultato dei due conteggi.

Procedendo "a cipolla", ci si libera fin dal principio dei compiti più elementari, e soprattutto si comincia subito a verificare che le strutture dati svolgano il loro compito correttamente: infatti, una volta scritte queste funzioni si potrà constatare a video che la scacchiera è correttamente costruita e che la valutazione della configurazione (2 pedine nere e 2 pedine bianche, dunque un pareggio) è corretta.

Terza fase (`othello3.c`) Si affronta quindi il corpo principale dell'algoritmo, cioè la lettura delle mosse da file di testo, la valutazione della loro correttezza e la loro esecuzione sulla scacchiera. Procedendo *top-down*, aggiungiamo

- l'apertura e chisura del file (`fopen` e `fclose`);
- il ciclo di lettura, `while`, a condizione iniziale, perché il file potrebbe anche essere vuoto, basato sul *parsing* del file una parola alla volta con la specifica `%s` e terminato quando la funzione `fscanf` non riconosce più un valore nello *stream*;
- la valutazione della mossa deve distinguere due casi:
  1. sospensione (mossa `--`): si deve valutare se il giocatore è veramente bloccato. Se lo è, non si deve fare nulla; altrimenti, si deve interrompere il programma con un messaggio di errore.
  2. mossa regolare (lettera e cifra): si deve ricavare dalla mossa gli indici della riga e colonna dove inserire la nuova pedina e valutare se la mossa è lecita. Se lo è, si deve eseguire la mossa; altrimenti, si deve interrompere il programma con un messaggio di errore.

Ovviamente, fare tutto questo è troppo pesante, per cui si sostituisce ogni operazione complessa con la chiamata (e quindi la dichiarazione e definizione) di un'apposita funzione. Via via che le mosse vengono lette, le stamperemo (`printf("%s\n",mossa);`) per verificare che la lettura sia corretta (poi cancelleremo la stampa, che non è richiesta). Infine, siccome queste operazioni richiedono di conoscere il giocatore corrente, definiremo una variabile (di tipo `char`) per conservare e aggiornare questa informazione, inizializzandola a `NERO`, e assegnandole ad ogni mossa il valore complementare a quello corrente.

Quarta fase (`othello4.c`) La scelta tra le due funzioni vuote ancora da definire è abbastanza arbitraria. Si potrebbe cominciare dalla più semplice, che è forse la verifica della sospensione, ma partiremo con l'esecuzione delle mosse, perché ha il vantaggio di poterne immediatamente valutare la correttezza eseguendola sui dati disponibili. Per eseguire una mossa bisogna prima verificare se sia lecita. Questo comporta anzi tutto che la casella sia libera; quindi, che la mossa porti a catturare pedine avversarie. Per verificare questa seconda condizione bisogna considerare la casella della scacchiera indicata nella mossa e scorrere ciascuna direzione orizzontale, verticale e diagonale. Nelle direzioni in cui la verifica ha esito positivo, si può eseguire fisicamente la mossa cambiando il colore delle pedine intermedie catturate. Una possibilità è definire una funzione che verifichi la mossa in una specifica direzione e una che la esegua; la seconda viene chiamata solo quando la prima ha dato esito positivo. Però le due funzioni sono molto simili, e il gioco richiede di eseguire la mossa in ogni direzione che consenta delle catture. Di conseguenza, preferiremo realizzare una sola funzione, che valuta la direzione data, in caso positivo esegue la mossa, e restituisce un valore logico che segnala se si sono catturate pedine avversarie oppure no. Per il momento, lasciamo questa funzione vuota, facendole restituire il valore `FALSE` come valore di *default*, da correggere in seguito, in modo che il codice sia sintatticamente corretto. Per scorrere le direzioni, una buona idea è di definirle, come in fisica, come vettori di spostamento, le cui componenti (sulle righe e sulle colonne) possono essere pari a -1, 0 o 1. Combinando i tre valori sulle due dimensioni, si ottengono 9 vettori, di cui quello interamente nullo va ovviamente scartato. Questo consente di scorrere le direzioni con un doppio ciclo e di indicare alle funzioni di verifica e di esecuzione il vettore di spostamento con una coppia di

componenti. Infine, bisogna tener traccia del fatto che almeno una direzione abbia dato esito positivo, per controllare che la mossa fosse effettivamente lecita.

Quinta fase (`othello5.c`) Nella fase seguente scendiamo al livello della verifica ed esecuzione delle mosse nelle singole direzioni. Dato il colore del giocatore che muove, si determina il colore dell'avversario. Si deve partire dalla casella immediatamente seguente a quella indicata dalla mossa, dunque passare da `(r,c)` a `(r+dr,c+dc)`. Da lì, ci si sposta, sempre nella direzione data verificando che esista una sequenza di caselle occupate dall'avversario. La sequenza termina quando:

1. si esce dalla scacchiera[1];
2. si trova una casella non occupata dall'avversario (dunque, libera o occupata dal giocatore).

La sequenza è lecita quando la casella finale:

1. è interna alla scacchiera;
2. è occupata dal giocatore;
3. non è immediatamente adiacente alla casella iniziale (in tal caso, la sequenza di caselle dell'avversario sarebbe vuota).

Sesta fase (`othello6.c`) Rimane solo da definire la funzione di verifica della sospensione. Questa deve scorrere tutte le caselle libere della scacchiera e, per ciascuna, verificare che tutte le direzioni sono non lecite per il giocatore. Si tratta quindi di quattro cicli annidati, due per le caselle e due per le direzioni. Se si trova una direzione lecita, si deve stampare un messaggio di errore e uscire.

Settima fase (`othello7.c`) A questo punto, un buon esercizio sia di programmazione sia di comprensione del concetto di struttura dati astratta è l'estrazione dal codice realizzato tutte le funzioni dedicate a gestire la scacchiera, in modo da costruire una libreria che consenta a qualsiasi programma di gestire una scacchiera senza conoscere nulla della sua effettiva implementazione. Il risultato di questa fase della lezione è il file `othello7.c`, con la libreria `scacchiera.c` e l'intestazione `scacchiera.h`.

1 Esercizio suggerito

Si generi una procedura che valuta la mossa migliore da eseguire in base al numero di pedine catturate. In realtà, questo principio è scorretto, dato che in genere conviene avere poche pedine impossibili da catturare che tante soggette a cattura. In caso di parità, alcune posizioni della scacchiera sono preferibili ad altre (ad esempio, gli angoli sono le posizioni migliori).

[1] Qui si potrebbe osservare che la verifica di appartenenza alla scacchiera potrebbe essere semplificata se si conoscesse la direzione in cui ci si sta muovendo: se si sta salendo verso l'alto, non occorre verificare che la riga sia minore di quella massima e non occorre verificare la colonna. Questo accelererebbe il codice, ma costringerebbe a scrivere otto versioni diverse della verifica. Anche se l'efficienza è in generale un criterio molto importante, in questo caso il programma è già abbastanza veloce da far preferire la semplicità. Nel progetto d'esame, sarà più probabilmente il contrario.