Lezione 5

L'esercizio illustra:

- la gestione dei vettori statici e dinamici (puntatori con memoria allocata nello *heap*);
- l'equivalenza fra vettori e puntatori;
- la differenza tra il passaggio dei parametri alle funzioni per valore e per indirizzo.

L'esercizio consiste nel caricare un elenco di numeri interi da un file di testo `sort.txt` in un vettore della lunghezza corretta e nello stampare il vettore stesso, interamente o in parte. Prelude a una lezione successiva, nella quale il vettore di numeri verrà ordinato.

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

Si scriva un programma `stampa_numeri.c` che apre un file di testo, carica nelle posizioni di 1 a n di un vettore dinamico della dimensione corretta i numeri in esso contenuti e stampa gli elementi del vettore nell'ordine.

Traccia della risoluzione

Per cominciare, faremo l'ipotesi (piuttosto forte) di sapere che il file di testo contiene esattamente `N` $= 100$ numeri interi. Il file `stampa_numeri0.c` contiene le solite inclusioni di librerie, le costanti simboliche `ROW_LENGTH` per la lunghezza del nome del file e `N` per il numero di interi da leggere, il `main` e la dichiarazione, chiamata e definizione della procedure per interpretare la linea di comando e ottenere il nome del file di testo. Queste ultime sono identiche a quelle realizzate nella lezione precedente.

Prima fase (`stampa_numeri1.c`) Impostiamo la risoluzione del problema in maniera *top-down*, dividendolo in due passi fondamentali:

1. caricamento dei numeri in un vettore statico;
2. stampa del vettore;

Entrambi i passi sono piuttosto semplici, e non richiedono nessuna osservazione particolare, salvo il fatto di osservare che entrambe le procedure ricevono non un vettore di interi, ma un puntatore a intero, come anticipato nei lucidi della lezione.

Seconda fase (`stampa_numeri2.c`) L'aspetto principale dell'esercizio sta nel fatto che molto spesso non conosceremo in anticipo (cioé mentre scriviamo il codice) il numero di elementi contenuti nel vettore, anche perché probabilmente vorremo utilizzare il programma per leggere file di lunghezza diversa. Questo richiede necessariamente di sostituire il vettore statico con un vettore dinamico, cioè con un puntatore a intero al quale venga assegnata un'area di memoria opportunamente dimensionata, creata con la funzione `calloc`, e deallocata al termine dell'esecuzione.

Sarebbe spontaneo ora dividere il problema in quattro passi, aggiungendo prima del caricamento dei numeri l'allocazione del vettore dinamico e dopo la stampa la deallocazione. Tuttavia, mentre la deallocazione va effettivamente aggiunta in coda al `main`, l'allocazione non può precedere il caricamento, dato che la lunghezza del vettore si può conoscere solo aprendo il file e contandone gli elementi[1]. Procederemo quindi ad aprire il file e a scorrerne gli elementi dal primo all'ultimo contandoli. Detto n il loro numero, allocheremo un vettore di $n + 1$ interi, per caricarli nelle posizioni da 1 a n secondo la convenzione di questo corso[2]. Quindi, torneremo al principio del file (`rewind`) e rileggeremo gli elementi caricandoli nel vettore. In questa seconda lettura, possiamo dare per scontato che sappiamo quanti sono e che abbiano il formato corretto.

Un altro aspetto importante di questo passo è che vettore e numero di elementi devono essere restituiti dalla procedura di caricamento come risultati. Siccome il risultato è multiplo (sono esattamente $n + 1$ interi), non si può usare l'istruzione `return` (in una lezione successiva, dedicata alle *tabelle*, vedremo un modo per farlo, il cui uso non è però molto comune). Bisogna invece passare i risultati come se fossero dei dati fittizii. La soluzione di questo passo fornita in rete é sbagliata: la lunghezza `n` e il vettore `V` vengono costruiti correttamente (si possono stampare da dentro la procedura), ma non vengono restituiti correttamente al `main`.

Terza fase (`stampa_numeri3.c`) L'errore della fase precedente è che i risultati vanno passati per indirizzo, e non per valore. Per la cardinalità n questo è facile: basta definire un parametro `int *pn`. Per il vettore si potrebbe obiettare che `int *V` é già un puntatore, come si è osservato nel primo passo dell'esercizio. Invece, non basta. Se l'allocazione del vettore V avviene dentro la funzione, il parametro formale `int *V` è solo una copia dell'omonimo puntatore `int *V` definito nel `main`:

- quando si alloca il nuovo vettore, la cella `V` della funzione di caricamento contiene l'indirizzo della nuova area allocata dinamicamente sullo *heap*, ma la cella `V` del `main` continua a contenere l'originale indirizzo casuale che aveva al principio del programma;
- quando si leggono i numeri contenuti nel file, essi vengono correttamente scritti nell'area allocata dinamicamente e puntata dal parametro `V` della funzione;

[1] Se fossimo fortunati, il file comincerebbe indicando il numero degli elementi, ma non siamo in questa situazione. Del resto, anche in questo caso, dovremmo comunque almeno aprire il file per leggere questo numero.

[2] In questo esercizio andrebbe bene anche indicizzarli da 0 a $n - 1$, ma se fosse necessario stampare oltre ai numeri anche degli indici da 1 a n la convenzione standard del C ci costringerebbe a ricordare ad ogni passo se per ottenere l'indice da stampare bisogna incrementare o decrementare quello effettivo nel vettore: ho visto intere generazioni di studenti consumare energie preziose in questo compito stupido, talvolta fallendo. C'è anche la possibilità di allocare un vettore dinamico con indici da 0 a $n-1$, spostarlo un passo indietro con l'*aritmetica dei puntatori* e usare il puntatore modificato. Questo andrebbe bene, ma richiederebbe di spostare nuovamente il puntatore un passo avanti prima di deallocarlo. Sforzo piuttosto ingiustificato per risparmiare una cella di memoria.

- quando si esce dalla funzione, la cella contenente il parametro `V` viene deallocata dallo *stack* e rimane solo la cella `V` del `main`, che punta un indirizzo a caso: l'area allocata dinamicamente è persa e irraggiungibile.

Per evitare questo errore, bisogna passare per indirizzo anche il puntatore `V`, cioè scrivere `CaricaVettoreInteri(char *filedati, int **pV, int *pn)`. Il nome `pV` del parametro cerca di ricordare che si tratta non di un vettore, ma di un puntatore alla cella `V` del `main` che è destinata a contenere l'indirizzo della prima cella del vettore dinamico. Per facilitare un po' l'interpretazione, si può introdurre un tipo ausiliario `vint` con l'istruzione `typedef int* vint;`. Questo nuovo tipo rappresenta i vettori dinamici di interi e coincide con i puntatori, ma si chiama in modo diverso per chiarire che ha un significato diverso. La dichiarazione diventa `CaricaVettoreInteri(char *filedati, vint *pV, int *pn)`.

A questo punto, la scrittura dell'allocazione del vettore e del caricamente dei dati diventa piuttosto barocca, perché il vettore da allocare e in cui scrivere non è `pV`, che è un puntatore a vettore, ma `*pV`, che è un vettore, e precisamente è un *alias* della variabile `V` del `main`. Quindi, dovremo scrivere ovunque `*pV`. Dovremo inoltre racchiuderlo spesso fra parentesi tonde per garantire la corretta precedenza degli operatori. Ad esempio, per leggere e scrivere i singoli elementi del vettore useremo l'espressione `(*pV)[i]`.

A questo punto, può venire il dubbio che anche la stampa del vettore richieda un puntatore al vettore stesso e uno alla sua lunghezza. Non lo richiede, per il semplice motivo che qui i due oggetti sono dati, non risultati, e la stampa non li modifica. Quindi si passano per valore, e il prototipo della funzione di stampa rimane `StampaVettoreInteri(vint V, int n)`. Anche funzioni che dovessero modificare gli elementi del vettore continuerebbero a richiedere di passare i parametri in questo modo. Infatti, pur essendo il vettore passato per valore, i suoi elementi sono implicitamente sempre passati per indirizzo. Il passaggio di vettori per indirizzo serve solo quando si vuole modificare il vettore in quanto oggetto globale (per esempio per riallocarlo, deallocarlo o spostarlo in altre aree di memoria, per esempio con l'aritmetica dei puntatori). Solo in questi casi occorre la complicata procedura sopra descritta.

A questo punto, è possibile stampare sottovettori del vettore sfruttando l'equivalenza con i puntatori,, cioè passando alla funzione di stampa non `V` e `n`, ma l'indirizzo della cella 0 del sottovettore e la sua lunghezza.

Del tutto banale, infine, é la deallocazione del vettore (`free`). Se fosse seguita da altre istruzioni, potrebbe valere la pena di annullare il puntatore (`V = NULL;`) per evitare di usarlo in maniera scorretta in seguito. Ma qui non ci sono altre operazioni. Si potrebbe anche pensare che non valga la pena di deallocare il vettore esplicitamente, perché viene fatto in automatico al termine del programma. In realtà, conviene farlo perché la deallocazione tipicamente produce situazioni di errore nei casi in cui il vettore è stato usato in modo scorretto durante l'esecuzione (per esempio, scrivendo in celle esterne ad esso). Questi errori non verrebbero scoperti lasciando la deallocazione al sistema operativo.