Lezione 7-8

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

- liste rappresentate con puntatori
- liste rappresentate con vettori e cursori

L'idea fondamentale è separare nettamente il concetto astratto di lista e la sua implementazione concreta. Questo comporta il dividere fisicamente da un lato la dichiarazione e definizione della struttura dati e delle operazioni che la manipolano (confinate in una libreria di due file) e dall'altro il loro uso (distribuito nel resto del programma). Questo comporta di usare sempre chiamate a funzioni per accedere ai dati, anche quando un accesso diretto sembrerebbe più semplice e naturale.

Il vantaggio di questo approccio è di poter gestire la struttura dati senza dover conoscere la sua implementazione concreta. È quindi persino possibile cambiare quest'ultima senza modificare il codice del programma, dato che cambia solo quello della libreria, che può essere sostituita in blocco da un'altra.

Questo approccio ha anche qualche svantaggio. Per prima cosa, avvicinarsi il più possibile a una struttura dati astratta con uno strumento concreto richiede un certo sforzo, molti dettagli tecnici di programmazione nel linguaggio C, e non è sempre possibile: a volte ci saranno comunque dissimiglianze tra la definizione teorica e l'implementazione concreta. Anche quando si riesce a mantenere una completa somiglianza, bisognerà introdurre qualche complicazione nella scrittura che a stretto rigore non sarebbe necessaria. In particolare, l'accesso ai dati tramite funzioni introduce per sua natura delle operazioni inutili (tutte quelle relative alla gestione del record di attivazione sullo *stack* di sistema). Alcune di queste inefficienze sono superabili arricchendo la libreria, cioè definendo qualche funzione aggiuntiva rispetto a quelle elencate nella definizione astratta. Altre sono superabili con strumenti tecnologici legati al linguaggio specifico utilizzato (per esempio, le macro in C o le funzioni *inline* in C++); di questi ultimi non parleremo.

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 vuole scrivere un programma `biblio.c` per la gestione per una biblioteca. Al principio, il programma acquisisce i libri disponibili in ordine alfabetico (prima per autore, poi per titolo). Quindi, il programma riceve un elenco di indicazioni di movimento dei libri, vale a dire dei prestiti e delle restituzioni:

- i libri prestati vanno rimossi subito dall'elenco dei libri disponibili e conservati in un elenco a parte;
- i libri resi non tornano immediatamente disponibili, perché il personale della biblioteca è insufficiente a riordinarli subito; vengono invece conservati al banco del prestito impilati uno sull'altro fino alla sera.

Il programma deve tener traccia dell'ordine delle restituzioni perché alla sera deve fornire al bibliotecario un elenco di istruzioni che lo aiutino a riordinare i libri: in particolare, i libri non hanno una posizione assoluta negli scaffali della biblioteca, ma stanno in un solo scaffale (che si immagina sufficientemente lungo) e vengono conservati in fila, sempre in ordine alfabetico. L'istruzione, quindi, indica per ogni

libro della pila (dalla cima al fondo) non una posizione assoluta, ma il libro già presente sullo scaffale prima del quale ciascun libro reso va riposto[1]. Si noti che quando un libro reso è stato riposto diventa immediatamente disponibile, e quindi può fare da punto di riferimento per l'inserimento dei successivi libri resi. Al termine, il programma deve stampare nuovamente la lista dei libri disponibili.

Formato di ingresso e di uscita L'elenco dei libri disponibili segue un formato molto semplice: ogni libro occupa una diversa riga, di lunghezza non superiore a 200 caratteri[2]. Sulla riga vengono stampati l'autore (cognome e iniziale del nome), una virgola e il titolo fra virgolette. Queste informazioni dettagliate non sono significative, perché il problema richiede di gestire l'intera riga in blocco, senza scomporla nelle informazioni dettagliate. I libri dell'elenco sono tutti diversi e sono disposti in ordine alfabetico per autore. Il termine dell'elenco è indicato da una riga contenente solo la parola `FINE`. Il file `scaffale01.txt` riporta un esempio di elenco dei libri disponibili.

```
Adams S., "Il principio di Dilbert"
Bertossi A., "Algoritmi e Strutture Dati"
Chaucer G., "I Racconti di Canterbury"
Hoeg P., "Il senso di Smilla per la neve"
Hofstadter D., "Godel, Escher, Bach"
Kernighan B. e Ritchie D., "Il linguaggio C"
King S., "L'ombra dello scorpione"
Pennac D., "La prosivendola"
Shakespeare W., "Le allegre comari di Windsor"
Yourcenar M., "L'opera al nero"
FINE
```

L'elenco dei movimenti avvenuti durante il giorno segue un formato simile: ogni movimento occupa una riga, costituita dalla parola chiave `PRESTITO` oppure `RESTITUZIONE`, seguita dal libro (nello stesso formato usato per l'elenco dei libri disponibili). I libri resi possono essere stati presi in prestito il giorno stesso (e quindi figurare nell'elenco iniziale) oppure in giorni precedenti (e quindi essere nuovi). L'elenco dei movimenti termina con una riga che contiene solo la parola chiave `FINE`. Il file `movimenti01.txt` riporta un esempio di elenco dei movimenti.

```
PRESTITO Bertossi A., "Algoritmi e Strutture Dati"
PRESTITO Kernighan B. e Ritchie D., "Il linguaggio C"
PRESTITO Adams S., "Il principio di Dilbert"
RESTITUZIONE Kernighan B. e Ritchie D., "Il linguaggio C"
PRESTITO Shakespeare W., "Le allegre comari di Windsor"
PRESTITO Pennac D., "La prosivendola"
PRESTITO Yourcenar M., "L'opera al nero"
RESTITUZIONE Bertossi A., "Algoritmi e Strutture Dati"
RESTITUZIONE Shakespeare W., "Le allegre comari di Windsor"
FINE
```

Per quanto riguarda i risultati, il programma deve restituire per prima cosa l'elenco dei libri ancora disponibili al termine della giornata, nello stesso formato

[1] Ovviamente, è una strana biblioteca, ma questo evita che l'esercizio si riduca ad accedere a un vettore con indici assoluti.

[2] Nel codice useremo stringhe da 256 per tener conto di a capi e terminatori e soprattutto del fatto che il file dei movimenti contien, oltre ai libri, anche le parole chiave `PRESTITO` e `RESTITUZIONE`. Si potrebbe essere precisi, ma ho preferito usare un valore arbitrario che fosse certamente sufficiente.

usato per acquisirlo al principio. Poi deve fornire una serie di istruzioni. Ogni istruzione consiste in una riga costituita dalla parola chiave `METTERE`, seguita dal libro reso, dalle parole chiave `PRIMA DI` e dal libro disponibile prima del quale il libro reso va riposto. Se il libro va messo per ultimo, si scriverà la parola chiave `METTERE` seguita dal libro e dalle parole chiave `PER ULTIMO`. Al termine delle operazioni, il programma stampa la parola chiave `FINE`. La stampa dell'elenco finale segue lo stesso formato usato per acquisire la lista iniziale.

Traccia della risoluzione

Prima di eseguire l'esercizio, conviene scorrere i lucidi fino a pagina 11, per acquisire il concetto di lista astratta e le basi delle implementazioni di liste con puntatori.

Modello del problema Per decidere come organizzare i dati del problema (e i risultati intermedi) occorre sapere quali operazioni bisogna compiere su di loro. Le operazioni richieste sono:

- scorrere i libri sullo scaffale per stamparne l'elenco;
- tenere traccia dei libri prestati e dei libri resi;
- determinare la posizione corretta di ciascun libro da riporre in base all'ordine alfabetico e all'elenco dei libri attualmente disponibili (cioè non prestati o già riposti in ordine) e fornire tali informazioni;
- scorrere nuovamente i libri sullo scaffale per stamparne l'elenco.

Occorre quindi gestire tre insiemi di libri: quelli sullo scaffale, quelli in prestito e quelli restituiti. Sono tre insiemi completamente dinamici, anche se esiste un sovrainsieme statico che li contiene tutti e di cui costituiscono una partizione (l'insieme di tutti i libri). A stretto rigore, questo insieme di tutti i libri potrebbe essere ricostruito scorrendo i due file di ingresso e riunendo tutti i libri che compaiono in almeno uno dei due. Quindi l'esercizio si potrebbe anche risolvere rappresentando i tre insiemi con tabelle, con una dimensione massima stimata a priori (magari per eccesso, sommando i libri inizialmente sullo scaffale e quelli soggetti a movimenti). Però fingeremo che il programma debba gestire in tempo reale la situazione, aggiornando via via la situazione, e che quindi durante la sua esecuzione non possa fare questa stima, come se il file dei movimenti non fosse interamente noto a priori, ma arrivasse un po' per volta. Questa ipotesi ci spinge a scegliere di rappresentare i tre insiemi con liste e a realizzare le operazioni prima elencate come operazioni su liste di libri.

In particolare, il primo insieme (libri sullo scaffale) è in ordine alfabetico e deve rimanerlo. Il secondo insieme (libri in prestito) è in genere privo di un ordine, ma si può anche pensare che sia ordinato in base all'ora del prestito (per esempio, se si volesse aggiungere una funzione di stampa della sequenza dei prestiti ancora aperti). Il terzo insieme (libri resi) è in un ordine determinato dalla sequenza di movimenti di restituzione: i libri sono impilati e i libri resi per primi stanno alla base della pila, quelli resi per ultimi alla cima. Inoltre, il fatto che i libri siano impilati restringe l'accesso all'ultimo libro aggiunto all'insieme, secondo la politica nota come *LIFO* (*Last In First Out*).

Punto di partenza (`biblio0.c`) Il punto di partenza dell'esercizio è il file `biblio0.c`, che già contiene:

1. le direttive per simulare il tipo `boolean`;
2. la costante simbolica `ROW_LENGTH` per gestire le righe in lettura e scrittura[3]
3. le variabili `file_libri` e `file_movimenti` dove conservare il nome dei file dei libri disponibili e dei movimenti effettuati;
4. la funzione `InterpretaLineaComando` che scrive i nomi dei due file nelle corrispondenti variabili recuperandoli dalla linea di comando (`argv[1]` e `argv[2]` rispettivamente).

Prima fase (`biblio1.c`) Nella prima fase della lezione, avviamo la realizzazione *top-down* dell'algoritmo. Decomponiamo il problema nelle seguenti operazioni:

1. creare la lista dei libri disponibili, quella dei libri prestati e quella dei libri resi: la prima è caricate da file, le altre due sono vuote;
2. eseguire i movimenti riportati nel file dei prestiti e dei resi;
3. stampare l'elenco dei libri disponibili;
4. stampare ed eseguire le istruzioni per riporre i libri resi;
5. stampare l'elenco dei libri disponibili;
6. distruggere le liste dei libri disponibili, prestati e resi (senza dimenticare gli elementi singoli delle liste stesse).

Le operazioni più sofisticate sono l'esecuzione dei movimenti e quella del riordino finale. La prima modifica tutte e tre le liste, la seconda le liste dei libri resi e disponibili. Siccome una funzione C non può restituire più risultati, passeremo le liste a queste funzioni per indirizzo.

Molte di queste operazioni suggeriscono la convenienza di avere a disposizione una libreria per la gestione di liste di libri. In effetti, è già fornito un abbozzo di libreria. Questa libreria propone un'implementazione a puntatori di una lista bidirezionale, circolare e dotata di sentinella. Questa implementazione consente un costo temporale molto basso per tutte le operazioni, al costo di un'occupazione di memoria maggiore (e un piccolo aumento nel costo temporale per alcune operazioni, dato che ci sono più puntatori da tenere aggiornati). La libreria comprende il file di intestazione `listalibri.h`, che dichiara:

- un `libro` come un vettore statico di caratteri;
- una `listalibri` come un puntatore a un `elemento` (dato che l'implementazione è a puntatori);
- una `posizione` come un puntatore a un `elemento` (dato che l'implementazione è a puntatori);

[3] Il testo indica una dimensione massima di 80 caratteri per ogni libro, a cui ne vanno sommati 12 per le parole chiave `PRESTITO` o `RESTITUZIONE` nel file dei movimenti, uno per lo spazio bianco fra la parola chiave e il libro, uno per l'a capo (se si legge ogni riga con la funzione `fgets`) e uno per il terminatore, per un totale di 94 caratteri. Ho preferito mantenere il valore 256 per andare sul sicuro e non introdurre complicazioni inutili.

- un `elemento` come una struttura contenente un libro e due posizioni: quella precedente e quella successiva (dato che l'implementazione è bidirezionale);
- tutte le funzioni elencate nei lucidi.

e il file listato `listalibri0.c`, che definisce le funzioni, ancora con i corpi vuoti. Per poter compilare senza messaggi di errore o avvertimento, alcune funzioni restituiscono valori fittizii convenzionali: `NO_LIST` per le liste, `NO_ELEMENT` per le posizioni, `TRUE` per i valori logici.

Un dettaglio tecnico di basso livello: implementando le liste con sentinella, i puntatori che le rappresentano rimangono sempre rigorosamente costanti (qualunque sia il contenuto della lista). Questo significa che le funzioni che modificano le liste potrebbero riceverle passate per valore, esattamente come si fa quando si passa una stringa a una funzione che ne modifica i caratteri o un vettore a una funzione che ne modifica gli elementi. Non lo faremo, perché non vogliamo legarci all'implementazione con sentinella, anche se la scrittura di tali funzioni ne risulterà un po' più complessa.

Stabilito di usare questa libreria, la prima fase dell'esercizio consiste nell'includere la libreria e aggiungere al codice la chiamata di opportune funzioni nel `main`, la loro dichiarazione fra i prototipi prima del `main` e la loro definizione vuota dopo il `main`. In particolare, la funzione che carica i libri disponibili si limita a creare e restituire una lista vuota.

Seconda fase (`listalibri.c`) Nella seconda fase, sospendiamo la modalità *top-down* per passare in modalità *bottom-up* e riempire la libreria di gestione delle liste dei libri. Questo si potrebbe fare anche in seguito (in effetti, potrebbe essere l'ultima fase dell'implementazione). La anticipiamo perché vogliamo avere al più presto dei risultati visibili, e siamo freschi della descrizione astratta delle operazioni su liste. Nell'ordine, definiamo:

- `listalibri crealista ()`, che alloca un elemento sentinella, lo fa puntare a sé stesso in entrambe le direzioni (lista circolare) e lo restituisce come valore della lista;
- `void leggelista (listalibri L, posizione p, libro TitoloAutore)`, che restituisce l'informazione associata a una data posizione di una data lista[4];
- `listalibri scrivelista (listalibri L, posizione p, libro TitoloAutore)`, che assegna un libro a una data posizione della lista[5];
- `boolean listavuota (listalibri L)`, che verifica se la lista è vuota: nel nostro caso, verificherà se la sentinella punta sé stessa.
  Come funzionerebbe in implementazioni prive di sentinella?
- `posizione primolista (listalibri L)`, che restituisce la prima posizione vera della lista, cioè un puntatore al successore della sentinella; se la lista è vuota, questo successore è la sentinella stessa.
  Come funzionerebbe in implementazioni prive di sentinella?

[4] Come anticipato, l'implementazione non rispecchia sempre perfettamente la struttura dati astratta. La funzione dovrebbe, infatti, restituire un libro. Però un libro è un vettore statico, e in C non si possono restituire vettori statici. Restituire un vettore dinamico vorrebbe dire allocare nuova memoria ad ogni lettura, probabilmente uno spreco. In generale, se il risultato di una funzione è ingombrante, è consigliabile passarlo per indirizzo. Nel caso del libro, che è un vettore di caratteri, passare per indirizzo i caratteri significa passare per valore il vettore. Ovviamente, sarebbe più efficiente accedere direttamente al dato, invece che copiarlo in un vettore passato dall'esterno, ma stiamo cercando di restare aderenti alla definizione astratta.

[5] Restituire la lista come risultato non sarebbe necessario, dato che la lista iniziale viene in effetti modificata. Lo facciamo per rimanere il più possibile fedeli alla definizione astratta.

- `posizione ultimolista (listalibri L)`, che restituisce l'ultima posizione vera della lista, cioè un puntatore al predecessore della sentinella; se la lista è vuota, questo predecessore è la sentinella stessa.
  Come funzionerebbe in implementazioni prive di sentinella?
  E in implementazioni monodirezionali?

- `posizione succlista (listalibri L, posizione p)`, che restituisce la posizione successiva lungo la lista; se la posizione è l'ultima, la posizione successiva è un puntatore alla sentinella.
  Come funzionerebbe in implementazioni prive di sentinella?

- `posizione predlista (listalibri L, posizione p)`, che restituisce la posizione precedente lungo la lista; se la posizione è la prima, la posizione precedente è un puntatore alla sentinella.
  Come funzionerebbe in implementazioni prive di sentinella?
  E in implementazioni monodirezionali?

- `boolean finelista (listalibri L, posizione p)`, che restituisce `FALSE` se la posizione è valida, `TRUE` se punta la sentinella.
  Come funzionerebbe in implementazioni prive di sentinella?

- `listalibri inslista (listalibri L, posizione p, libro TitoloAutore)`, che alloca un nuovo elemento, copiandovi il libro indicato, e lo inserisce nella lista nella posizione `p`, cioè spingendo in avanti l'elemento in posizione `p` e i successori e lasciando dove sono tutti i predecessori; infine, restituisce la lista modificata[6];
  Come funzionerebbe in implementazioni prive di sentinella?
  E in implementazioni monodirezionali?

- `listalibri canclista (listalibri L, posizione *pp)`, che sgancia l'elemento in posizione `*pp` dalla lista ricollegando il precedente e il successivo e lo dealloca, restituendo la lista modificata[7];
  Come funzionerebbe in implementazioni prive di sentinella?
  E in implementazioni monodirezionali?

- `void distruggelista (listalibri *pL)`, che distrugge interamente la lista, eseguendo `canclista` sul suo primo elemento finché la lista non rimane vuota, poi dealloca la sentinella e infine assegna a `L` il valore fittizio `NO_LIST` per indicare che non è più una vera lista.

[6] Un dettaglio tecnico potenzialmente insidioso: passare la posizione `p` per valore significa che al termine il relativo puntatore indica in realtà la posizione $p + 1$. Qui si è deciso di ignorare la cosa, per semplificare l'implementazione e per consentire di passare alla funzione `inslista` come argomento non solo variabili di tipo `posizione`, ma anche funzioni che restituiscano posizioni, come `primolista` o `succlista`, che non potrebbero essere passate per indirizzo, perché il risultato di una funzione non ha un indirizzo stabile.

[7] Qui si presenta lo stesso problema tecnico descritto per l'inserimento riguardo la posizione: se si passa la posizione per valore, il relativo puntatore continua a indicare l'elemento cancellato. Nella definizione astratta, invece, l'elemento successivo risale a occupare la posizione dell'elemento cancellato. Qui si è deciso di non ignorare la cosa, perché avere un puntatore a un elemento deallocato è spesso pericoloso. Quindi la posizione viene passata per indirizzo e la procedura modifica il puntatore in modo che punti l'elemento corretto. Non sono soddisfatto di questa incoerenza fra inserimento e cancellazione, ma non riesco a decidermi per alcuna delle due soluzioni estreme (passaggio per valore o per indirizzo in entrambi i casi): la prima è una probabile fonte di errori, la seconda è complicata, diversa dalla definizione astratta e impedisce di passare come argomento alle due funzioni il risultato di una funzione.

Terza fase (`biblio2.c`) Ora torniamo a seguire l'approccio *top-down*, realizzando la procedura `CaricaLibri`, che legge dal primo file di ingresso una riga alla volta e riconosce la prima parola della riga (consiglio: `sscanf` e `strcmp`): se è la parola chiave di terminazione `FINE`, interrompe la lettura; altrimenti, inserisce il nuovo libro nella lista di quelli disponibili. La lettura della riga comporta di risolvere una tecnicalità non banale che è descritta in dettaglio nell'ultima sezione perché non riguarda gli algoritmi, ma la programmazione in C. Qui osserviamo solo che bisogna scegliere se leggere l'intera riga escluso l'a capo finale e saltarlo per passare alla riga successiva, oppure leggere la riga compreso l'a capo. In questo caso, bisogna poi scegliere se cancellarlo in qualche modo o conservarlo (il che influenza le operazioni di stampa e confronto fra libri e di stampa delle operazioni). Il nuovo libro va inserito nella posizione finale, che non è l'ultima occupata, ma quella seguente. Questa posizione non è direttamente accessibile con le funzioni della libreria: per ottenerla, possiamo applicare `succlista` al risultato di `ultimolista`. Ne risulta un'espressione abbastanza barocca da suggerire che possa essere utile aggiungere alla libreria una funzione `accodalista` per inserire un nuovo elemento in coda a una lista chiamando una sola funzione anziché tre. Non lo faremo, ma è un esercizio consigliabile.

Sempre in questa fase, realizziamo anche una funzione `StampaLibri`, che ci permette di verificare almeno in parte la correttezza semantica delle procedure fin qui realizzate. Si tratta di scorrere la lista con una variabile di tipo `posizione`, partendo da `primolista`, leggendo ogni elemento con `leggelista` e passando al successivo con `succlista`. Anche questa stampa non è molto efficiente, dato che la lettura copia la stringa in un'altra, che viene poi stampata. Per renderla efficiente si potrebbe aggiungere alla libreria una funzione `stampalista` che essendo interna alla libreria stessa, avrebbe il permesso di accedere direttamente al dato.

Quarta fase (`biblio3.c`) Passiamo all'esecuzione dei movimenti, che si immagina avvenga in tempo reale durante la giornata, via via che nuovi movimenti vengono comunicati e nuove righe si aggiungono in fondo al relativo file. La procedura `EsegueMovimenti`:

- apre il file dei movimenti;
- legge ciascuna riga e ne separa la prima parola dal rimanente; in questo caso, la riga può essere letta con o senza l'a capo finale; la funzione `sscanf` permetterà di separare il comando dal resto (se c'è un resto) con la specifica `"%s %[^\n]"`, dove `"%s"` riconosce la prima parola e `"%[^\n]"` tutto ciò che segue escluso l'a capo finale[8];
- in caso di prestito:
  1. cerca il libro sullo scaffale, chiamando un'apposita funzione `TrovaLibro`, che per ora dichiariamo soltanto, obbedendo all'approccio *top-down*,
  2. lo cancella dalla lista dei libri disponibili,
  3. lo inserisce nella lista dei prestiti (*in quale posizione?*);
- in caso di resa:
  1. cerca il libro nella lista dei prestiti,
  2. lo cancella dalla lista dei libri disponibili,

[8] Si noti lo spazio fra le due specifiche, che è importante, perché serve a saltare gli spazi compresi fra il comando e il titolo del libro.

3. lo inserisce nella lista dei resi in posizione iniziale.

Una possibile considerazione riguarda il fatto che per spostare un libro da una lista a un'altra, applichiamo in sequenza la cancellazione alla prima lista e l'inserimento nella seconda, con una deallocazione e un'allocazione di memoria dinamica, che potrebbero facilmente essere risparmiate se ci limitassimo ad aggiornare i puntatori dell'elemento che contiene il libro in modo da sganciarlo dalla prima lista e agganciarlo alla seconda. Questo è impossibile con la libreria nella sua forma base, ma potremmo aggiungere una funzione `sposta_lista` che esegua l'operazione nel modo più efficiente. È un altro esercizio consigliabile[9].

Per concludere, realizziamo anche la funzione `TrovaLibro`. Dato che scorre una lista, ha una forte somiglianza con la procedura di stampa, e anch'essa si candida naturalmente ad essere inclusa nella libreria di gestione della lista per evitare le inefficienze associate all'esigenza di gestire la struttura dati in modo astratto.

Quinta fase (`biblio4.c`) Ora procediamo al riordino e alla stampa a video delle istruzioni corrispondenti. La procedura `RiordinaResi`:

- scorre la lista dei libri resi dalla cima, estraendo e cancellando via via il primo;
- per ciascun libro, scorre la lista dei libri disponibili cercando il primo libro alfabeticamente successivo a quello corrente[10];
- se lo trova, inserisce il libro reso prima di esso e stampa il corrispondente messaggio a video; altrimenti, lo inserisce in coda e stampa il corrispondente messaggio a video.

Anche qui, lo spostamento di un libro da una lista a un altro verrà eseguito cancellandolo dalla prima e inserendolo nella seconda, a meno che non si sia deciso di approfondire l'esercizio arricchendo la libreria con la procedura `sposta_lista`.

Sesta fase (`listalibri-cursori.c`) Ora si può procedere a creare una nuova libreria di gestione, basata sull'implementazione con cursori, ma ancora bidirezionale, circolare e con sentinella. Ovviamente, ciascuna delle altre 14 implementazioni presentate in precedenza (7 a puntatori e 7 a cursori) costituisce un'utile esercizio.

La differenze fondamentali rispetto all'implementazione a puntatori sono:

- la lista è ospitata in un vettore allocato dalla funzione `crealista`;
- i puntatori sono sostituiti da indici numerici: l'elemento in posizione `p` della lista `L` è banalmente `L[p]`; per accedere agli elementi, bisogna quindi conoscere sia la posizione sia la lista;
- la creazione della lista richiede l'allocazione del vettore[11], l'impostazione della sentinella, che punta sé stessa, e la costruzione di una lista libera contenente tutti gli altri elementi;

[9] Va notato che la funzione è piuttosto semplice se sposta elementi da una lista a un'altra, ma potrebbe venire la tentazione di usare la stessa funzione per spostare un elemento da una posizione a un'altra nella stessa lista. Vi sono casi in cui questo potrebbe portare a errori nella gestione dei puntatori. Le soluzioni sono due: vietare lo spostamento interno a una lista (con un controllo esplicito) oppure implementare la funzione con molta attenzione, in modo che sia corretta anche nei casi più insidiosi.

[10] Questo problema è molto simile alla ricerca esatta di un libro in una lista.

[11] Il vettore è allocato con una dimensione costante `LIST_SIZE+2`, che tiene conto degli elementi massimi ospitabili, della sentinella (in posizione `SENTINELLA`) e di una sentinella aggiuntiva (in posizione `SENTINELLA_LIBERA`) per la così detta *lista libera*, che ospita gli elementi del vettore non impiegati nella lista data). L'allocazione è dinamica, anche se con una dimensione costante. Questo è poco elegante, ma necessario per mantenere l'astrazione, dato che alcune funzioni della libreria restituiscono liste, e in C una funzione non può restituire un vettore statico.

- inserimenti e cancellazioni corrispondono a spostamenti fra lista vera e propria e lista libera: ci sono più operazioni sulle posizioni, ma vengono a mancare le allocazioni e deallocazioni.
- la funzione `inslista` deve verificare che la lista libera non sia vuota; se lo fosse, l'inserimento non potrebbe avvenire (contrariamente al caso delle liste a puntatori, dove l'inserimento fallisce solo quando la memoria dinamica è esaurita). Bisogna decidere come gestire questa situazione: analogamente al caso delle tabelle, si può terminare il programma (approccio scelto nelle soluzioni in rete), ignorare l'inserimento, ignorarlo segnalando il problema alla funzione chiamante, o riallocare il vettore con una dimensione maggiore, tipicamente doppia[12]

La nuova implementazione è piuttosto diversa dalla precedente, ma non richiede alcuna modifica al file `biblio.c`: potenza delle strutture dati astratte, salvo (dettaglio da non dimenticare!) di includere l'intestazione della nuova libreria anziché quella della vecchia e di compilare la nuova libreria anziché la vecchia. Il risultato di questa fase della lezione sono i file `listalibri-cursori.h` e `listalibri-cursori.c`.

Ospitare più liste disgiunte su un solo vettore Rigorosamente parlando, è anche possibile gestire tutte e tre le liste su un solo vettore, dato che sono mutuamente esclusive. Tecnicamente, questo richiede 4 posizioni aggiuntive per le sentinelle delle tre liste e la sentinella della lista libera. Anzi, osservando che ogni elemento appartiene a una delle tre liste (si tratta in effetti di una partizione), non occorre nemmeno definire una lista libera. All'indubbio vantaggio in termini di spazio fa da contrasto lo svantaggio che le dichiarazioni e le definizioni riportate nella libreria diventerebbero specifiche per il caso particolare della partizione di un insieme dato in tre liste. Si tratta valutare se la maggior specificità è compensata dalla maggiore efficienza. Va anche detto che ospitare più liste su un solo vettore confligge con la struttura dati astratta che abbiamo definito e usato finora. Occorrerebbe in effetti definire una struttura dati astratta per la gestione di una pluralità di liste disgiunte.

Leggere una riga di testo (un sordido problema informatico) Nella terza fase dell'esercizio, si pone il problema di leggere una riga del file di ingresso, decidendo se:

1. non includervi l'a capo finale,
2. includerlo e subito cancellarlo,
3. includerlo e basta.

La terza soluzione è la più semplice: basta applicare la funzione `fgets`. Però bisogna ricordare nel seguito che i libri terminano con un a capo. Questo influisce sulle successive operazioni di stampa (ogni libro andrà automaticamente su una riga diversa, senza indicarlo esplicitamente) e di confronto (l'a capo compare in tutti i libri letti in questo modo, per cui non dovrebbero esserci problemi). Soprattutto, crea problemi alla stampa delle istruzioni di riordino, nelle quali si stampa il libro reso seguito dalla posizione in cui inserirlo e tutto questo andrebbe fatto su una sola riga. Questo tende a escludere tale soluzione.

[12] In questo caso, bisogna anche inserire i nuovi elementi nella lista libera.

Nelle prime edizioni del corso ho applicato la seconda soluzione, applicando la funzione `fgets` per leggere la riga compreso l'a capo e poi cancellando quest'ultimo. Per troncare una stringa in qualsiasi posizione basta sovrascrivere il primo carattere che va escluso con il terminatore `'\0'`. In questo caso, il carattere è `'\n'`) e sta in ultima posizione, per cui si può applicare l'istruzione `Riga[strlen(Riga-1)] = '\0';` e quasi sempre non si avranno problemi[13]. Purtroppo, sistemi operativi diversi gestiscono in modo diverso l'andare a capo: alcuni lo rappresentano con un singolo carattere, altri con due. Bisogna ricordare le macchine da scrivere meccaniche, che andavano a capo eseguendo due movimenti: tirare il carrello verso sinistra e poi ruotare il tamburo per spostare il foglio di carta. I compilatori C dovrebbero adeguarsi tutti in modo che il programmatore possa ragionare sempre in termini di un singolo carattere.

Purtroppo, se si apre con un sistema operativo un file di testo creato con un altro e non riadattato durante il trasferimento, sono possibili problemi. In particolare, gli esempi `scaffale01.txt` e `movimenti01.txt` sono stati creati sotto Windows, e ogni riga termina con i caratteri `'\r'` (rientro del carrello) e `'\n'` (passaggio alla riga seguente). Usandoli sotto altri sistemi operativi senza una traduzione, possono ingannare il compilatore C, che cancellerà solo `'\n'`, lasciando `'\r'` che crea nella stampa un effetto bizzarro di rientro al principio della riga e conseguente sovrascrittura. Il tutto è ben spiegabile solo alla luce del meccanismo. Una soluzione pratica è adattare i file al nuovo sistema con i seguenti comandi da terminale:

```
sed -i 's/\r//g' scaffale01.txt
sed -i 's/\r//g' movimenti01.txt
```

Probabilmente, funziona anche aprire i file, e cancellare e ricreare gli a capi a mano uno per uno.

Passando alla prima soluzione, essa consiste nel leggere la riga con l'istruzione `fscanf(fp,"%[^\n]\n",Riga);` dove la specifica `"%[^\n]"` riconosce qualsiasi sequenza di caratteri che non (`^`) includa l'a capo, mentre il termine `\n` finale legge l'a capo (in effetti, legge qualsiasi sequenza di separatori: a capi, spazi bianchi e tabulazioni) senza scriverlo da nessuna parte. Non ho testato questa soluzione su altri sistemi operativi, quindi invito a farmi sapere se dà luogo a problemi.

[13] Un problema di questa istruzione è che, applicata su una riga vuota, cioè di lunghezza nulla, va a scrivere nella posizione `Riga[-1];` provocando certamente problemi, ma le righe lette da `fgets` dovrebbero essere tutte non vuote.