

Esercitazioni di Tecniche di Programmazione

3. Terza esercitazione autoguidata: progetto gestione voli

due avvertenze:

- 1) Le soluzioni agli esercizi, le versioni di programmi dati nel testo delle esercitazioni e quant'altro sono raggiungibili tramite la pagina web del corso. Nel titolo di ogni sezione è specificato tra parentesi il nome del (o dei) file in cui è proposta una soluzione (se disponibile ...).
- 2) I programmi che scriveremo dovranno essere in accordo con la definizione standard **ANSI C** del linguaggio C; perciò, prima di cominciare vogliamo assicurarci che l'ambiente di programmazione che usiamo "usi" anche lui la medesima definizione.
 - a. SE si usa il Dev C++, nella versione 4.9.9.2 (lingua inglese) bisogna andare nel menu' "Tools", selezionare "Compiler Options", scegliere "Settings" e poi "C Compiler" (selezionare almeno "Support all ANSI Standard C Programs")
 - b. Se si usa il vecchio ed eroico Turbo C++, per essere sicuri di star usando ANSI C, bisogna assegnare opportunamente una certa opzione: aprire il menù OPTIONS, selezionare *Compiler* e poi *Source*. Nella finestra di scelta che appare, selezionate ANSI C.

Qui ci occupiamo di scrivere un programma abbastanza lungo, dedicato alla gestione di una tabella di dati. Una tabella e' una struttura dati che rappresenta all'interno di un programma, un archivio di dati del mondo reale.

3.1. Cos'e' una tabella (in breve...)

Di solito la tabella viene definita per permettere la gestione automatizzata di una collezione di dati strutturati (dati composti da varie informazioni di tipo disomogeneo e quindi normalmente rappresentati mediante `struct`).

Un esempio di tabella e' l'elenco telefonico (una collezione di elementi; ciascuno elemento e' l'insieme di dati relativi ad una persona, come il nome, l'indirizzo, il numero di telefono, titoli professionali etc...).

Un altro esempio e' l'elenco degli iscritti ad una associazione (anche qui ogni elemento in tabella e' una collezione di dati relativi ad una persona, come il nome e indirizzo, ma anche l'anzianita' d'iscrizione, la quota associativa pagata etc...).

Le funzioni generiche di gestione di una tabella sono

- l'aggiunta di un elemento nella tabella,
- l'eliminazione di un certo elemento,
- la modifica dei dati memorizzati in un certo elemento
- e la ricerca di un certo elemento tra quelli memorizzati nella tabella.

Di solito uno dei dati memorizzati in ciascun elemento lo caratterizza univocamente e permette di distinguere quell'elemento da tutti gli altri: ad esempio il codice fiscale permette di specificare una certa persona; la matricola permette di indicare univocamente uno degli associati al club, etc...

Questi "dati caratteristici" vengono denominati chiavi di ricerca, o piu' brevemente, *chiavi*, e vengono usati quando si stanno effettuando operazioni di ricerca e selezione di un certo elemento nella tabella.

3.2. Gestione voli (primo passo VOLI1.C)

Il programma deve permettere la gestione di un archivio contenente informazioni sui voli aerei di un certo giorno.

Un volo e' caratterizzato da

- codice (5 caratteri alfanumerici)
- destinazione (stringa di caratteri)
- ora di partenza (ore, minuti: due interi)
- numero di posti attualmente liberi (un intero)

La tabella dei voli di quel giorno e' una collezione di voli (informazioni su voli) su cui e' possibile eseguire le seguenti operazioni di gestione:

- stampare un volo della tabella, caratterizzato da un certo codice
- stampare i voli della tabella
- aggiungere un volo alla tabella
- eliminare un volo avente caratterizzato da un certo codice
- modificare l'ora di partenza di un volo caratterizzato da un certo codice
- prenotare k posti in un volo caratterizzato da un certo codice (cioe' modificarne il numero di posti liberi, diminuendolo, se possibile, di k)
- memorizzare i dati di una tabella di voli in un file di voli
- caricare nella tabella dati da un file di voli

Si vede che in questa tabella il campo codice e' la "chiave di ricerca" per i voli memorizzati.

Scrivere un abbozzo del programma che dovra' permettere tutte le operazioni sopra indicate. In questo abbozzo di programma ci limitiamo a fornire

- la definizione della struttura dati che useremo per rappresentare la tabella dei voli;
- la dichiarazione e implementazione (programmazione) delle funzioni necessarie a realizzare le prime tre operazioni elencate sopra;
- una funzione main in cui venga stampato un menu' di scelte (stampa di un volo, stampa dei voli in tabella, aggiunta di un volo in tabella, termine programma) e vengano conseguentemente usate le funzioni definite.

Il programma va scritto e testato. Una proposta di soluzione e' in VOLI1.C.

Suggerimento 1di3: (sulle strutture dati, tratto da VOLI1.C)

tabelle di 10 voli, per scrivere e testare il programma. Dopo potremo cambiare questa dimensione

```
#define MAXVOLI 10
```

il tipo struct per rappresentare un volo contiene un campo oraPartenza. L'ora e' ben rappresentabile con un'altra struct

```
struct ora {
    int ore, minuti;
};

struct volo {
    char codice[6];
    char * destinazione;
    struct ora oraPartenza;
    int postiLiberi;
};
```

per i voli, decidiamo di definire un sinonimo di tipo

```
typedef struct volo TipoVolo;
```

/* la tabella di voli consiste di un array di MAXVOLI voli, ma anche di un intero quantiVoli che in ogni momento specifica quanti sono i voli effettivamente presenti in tabella. Faremo in modo che, se in un dato momento ci sono k voli in tabella, essi siano i primi k elementi dell'array e quantiVoli sia uguale a k.

Ogni scansione della tabella si limitera' agli elementi di indice da 0 a quantiVoli-1.

Quindi una tabella di voli e' una struttura a due campi.

```
*/
```

```
typedef
    struct {
        TipoVolo arrayVoli[MAXVOLI]; /* sostegno di memoria */
        int quantiVoli; /* quanti voli presenti nel sostegno */
    }
TipoTabella;
```

Suggerimento 2di3: ipotesi su alcune funzioni che useremo (solo intestazioni)

```
void stampaTabella(TipoTabella t);  
/* stampa tutti i voli della tabella */  
  
/* !!! */  
/* per entrambe le precedenti funzioni sara' comodo usare una  
funzione come la seguente ... */  
void stampaVolo(TipoVolo v);  
/* stampa il volo v */
```

```
int aggiungiVolo(TipoTabella *t);  
/* aggiunge un nuovo volo nella tabella *t, chiedendo e leggendo  
opportunamente i dati relativi
```

Restituisce 1 o 0 a seconda della riuscita dell'aggiunta.

Passiamo l'indirizzo **t** della tabella da modificare, in modo che le modifiche vengano fatte direttamente sulla tabella *t (e non su una sua copia).

Sia (*t).arrayVoli (nell'elemento di indice dipendente da quantiVoli), che (*t).quantiVoli saranno modificati.

L'elemento in cui si inserisce il nuovo volo e'

(*t).arrayVoli[KKK]

dove k deve essere uguale al campo quantiVoli della tabella (KKK=(*t).quantiVoli).

Poi (*t).quantiVoli deve crescere di uno, perche' abbiamo aggiunto un volo in tabella.

*/

Suggerimento 3di3: ecco la main()

```
int main() {
    TipoTabella tabVoli;
    int riuscita,
        scelta;          /* scelta nel menu' */
    char buffer[40];     /* per leggere stringhe */

    tabVoli.quantivoli=0; /* inizializzazione del numero di
                           voli presenti in tabella (cosa ci sia
                           effettivamente in tabVoli.arrayVoli
                           e' di poco interesse. Tanto le
                           scansioni che faremo saranno sempre
                           limitate da quantivoli*/

    do {
        /* STAMPA MENU' */
        printf(" - scegli -\n");
        printf(" - stampa dei voli (1) -\n");
        printf(" - stampa di un certo volo (2) -\n");
        printf(" - aggiunta di un volo (3) -\n");
        printf(" - fine (0) -\n");

        scanf("%d", &scelta); /* SCELTA ESPRESSA DALL'UTENTE */

        switch (scelta) { /* ELABORAZIONE IN BASE ALLA SCELTA */
            case 1:
                printf("- %d voli in tabella:\n", tabVoli.quantivoli);
                stampaTabella(tabVoli);
                break;
            case 2:
                /* funzione non ancora implementata;
                   (rimanadata al prossimo esercizio)
                printf(" - codice volo? ");
                scanf("%s", buffer);
                stampaQuelVolo(tabVoli, buffer);
                */
                break;
            case 3:
                riuscita=aggiungiVolo(&tabVoli);
                if(!riuscita)
                printf(" - aggiunta non effettuata -\n");
                else
                printf(" - fatto -\n");
                break;
            case 0:
                printf(" - USCITA DAL PROGRAMMA\n");
                break;
            default:
                printf(" - opzione sballata\n");
        } /* fine switch */
    } while (scelta!=0); /* fine do_while*/

    printf("\nFINE\n");
    return 0;
}
```

3.3. Gestione voli (secondo passo VOLI2.C)

Incrementare il programma costruito al passo precedente, con la funzionalita' di stampa di un volo, dato il codice. (**E poi testarlo ...**)

Suggerimento 1di2:

La funzione da implementare e' la seguente:

```
void stampaQuelVolo(TipoTabella t, char cod[]);
```

Suggerimento 2di2:

Pero' per stampare il volo di codice cod, bisogna cercarlo nella tabella. Questo problema di cercare un volo di codice dato ricorre, verosimilmente spesso nel resto del programma. Ecco come risolvere il problema una volta per tutte, con una funzione:

```
int indiceVolo(TipoTabella t, char cod[]);  
/* cerca il volo di codice cod in t;  
cioe' cerca nell'array t.arrayVoli, limitando la scansione agli  
elementi che vanno da indice 0 a indice t.quantivoli-1  
  
restituisce l'indice del volo oppure, se non lo ha trovato, -1  
*/
```

3.4. Gestione voli (terzo passo VOLI3.C)

Incrementare il programma costruito al passo precedente, con la funzionalità di eliminazione di un volo di codice dato. (E poi testarlo ...).

Suggerimento 1di2:

Bisogna realizzare la funzione di eliminazione ed estendere la main() in modo che permetta di scegliere anche l'opzione di eliminazione. Quando viene scelta questa opzione, il programma chiede il codice del volo da eliminare e poi chiama la funzione di eliminazione.

La funzione da implementare è la seguente:

```
int eliminaVolo(TipoTabella *t, char cod[]);
/* elimina il volo di codice cod dalla tabella *t
(cambiano arrayVoli e quantiVoli)

restituisce 0 o 1 a seconda del successo dell'operazione

Si cerca l'elemento da eliminare, assegnando ad un indice il
valore ritornato dalla funzione di ricerca,
ad esempio
    k=indiceVolo(*t, cod);
e
lo si ricopre copiandoci sopra l'ultimo elemento della tabella;
ad esempio
    t->arrayVoli[k] = t->arrayVoli[ultimo];
(quanto vale l'indice "ultimo"?
poi si decrementa quantiVoli
ad esempio
    t->quantiVoli-=1;
*/
```

Suggerimento 2di2:

```
ultimo=t->quantiVoli-1;
```

3.5. Gestione voli (quarto passo VOLI4.C)

Incrementare il programma costruito al passo precedente, con le funzionalità di modifica richieste, cioè cambio dell'ora di partenza di un volo di codice dato e prenotazione di un certo numero k di posti su un volo di codice dato. (E poi testarlo ...).

Bisogna realizzare due funzioni apposite,

```
cambiaOraPartenza(...) e  
cambiaPostiLiberi(...);
```

e poi modificare la main() in modo che anche queste nuove operazioni siano accessibili dal menu'.

Suggerimento: le funzioni da realizzare potrebbero essere le seguenti:

```
int cambiaOraPartenza (TipoTabella *t, char cod[], int nuovaOra,  
int nuoviMin);  
/* restituisce 0 o 1 a seconda del successo dell'operazione */  
  
int cambiaPostiLiberi(TipoTabella *t, char cod[], int k);  
/* diminuisce di k i posti liberi, SE POSSIBILE  
restituisce 0 o 1 a seconda del successo dell'operazione */
```

Il parametro tabella qui deve essere l'indirizzo della tabella "attuale" da modificare...

3.6. Gestione voli (quinto passo VOLI5.C)

Incrementare il programma costruito al passo precedente, con le funzionalità di salvataggio su file della tabella e caricamento da file della tabella. (E poi testarlo ...).

Bisogna realizzare due funzioni apposite,

```
void daTabellaInFile(TipoTabella t, char *nmf);
```

```
/*
```

```
riceve
```

- una tabella da scaricare in memoria secondaria e
- il nome del file in cui memorizzare le informazioni;

```
e scarica i dati della tabella nel file di nome nmf*/
```

```
void daFileInTabella(char *nmf, TipoTabella *t);
```

```
/* riceve
```

- l'indirizzo di una tabella in cui memorizzare i dati contenuti in un file e
- il nome del file da cui trarre i dati per riempire la tabella;

```
e carica nella tabella puntata da t (cioe' *t) i voli  
contenuti nel file di nome nmf */
```

e poi modificare la main() in modo che anche queste nuove operazioni siano accessibili dal menu'.

3.7. Gestione voli (sesto passo VOLI6.C)

Qui si richiede di risolvere il problema della gestione dei voli con un approccio piu' dinamico, cioe' facendo in modo che la memoria occupata dall'array di voli arrayVoli (campo della tabella TipoVoli) non occupi un numero fissato MAXVOLI di locazioni di tipo TipoVolo (struct) ma modifichi l'occupazione di memoria in ragione delle necessita' (cioe' del numero di voli effettivamente memorizzati nella tabella).

Per realizzare questo scopo si suggerisce di adottare il seguente schema:

- il tipo TipoTabella viene modificato, in modo che il campo arrayVolo non sia piu' un array statico di MAXVOLI voli, ma il puntatore ad un array di voli dinamicamente allocato:
TipoVolo * arrayVoli;
- **l'inizializzazione** della tabella non si limita piu' all'azzeramento del campo .quantivoli, ma deve anche procedere ad una prima allocazione del blocco di voli .arrayVoli. Questa allocazione fa in modo che arrayVoli punti ad un blocco di MAXVOLI voli (dove MAXVOLI e' un simbolo di costante definito opportunamente, per il quale si consiglia un valore abbastanza basso, es 5 o 6);
- la **funzione di caricamento** dei dati da file a tabella viene modificata, in modo che venga
 - o deallocato il blocco eventualmente puntato da arrayVoli e
 - o allocato (e assegnato ad arrayVoli) un blocco di voli che verra' riempito con i dati contenuti nel file; in particolare
 - il primo dato letto dal file dice quanti sono i voli presenti nel file;
 - il blocco viene allocato con un numero di voli di circa il 30% superiore a quelli effettivamente necessari, cosi' dopo il caricamento dei dati, ci sara' spazio per aggiungere nella tabella nuovi voli, ma senza esagerare con lo spazio occupato;
- La **funzione di aggiunta** di un nuovo volo in tabella viene cambiata in modo che quando l'aggiunta non e' possibile per mancanza di spazio in arrayVoli, anziche' lasciar perdere
 - o si fa in modo che arrayVoli punti ad un blocco di voli "esteso" (un po' piu' grande, ad esempio del 30%)
 - o e poi si aggiunge il volo come si faceva prima

Per fare in modo che arrayVoli punti ad un blocco di voli che contiene i voli gia' presenti in tabella piu' qualche altro spazio per nuovi voli, si possono usare varie soluzioni. La piu' immediata consiste nel

- o salvare la tabella in un file TEMP.TXT
- o ricaricare subito la tabella, con la funzione di caricamento modificata, discussa in un punto precedente.

Si consiglia di procedere facendo una copia del file in cui era stato risolto completamente il punto precedente (la cui proposta di soluzione era in VOLI5.C) e modificare questa copia, seguendo l'ordine delle suggerimenti elencati sopra;

Durante questa opera ci si potrebbe rendere conto che, a proposito di modifiche alla struttura dati di Tabella dei Voli, non tutto e' stato detto fin qui: potrebbe essere necessario aggiungere un'altra modifica.

La scoperta di questa modifica e' bella se viene fatta autonomamente ... ma comunque se ne parla nel prossimo suggerimento.

Suggerimento:

visto che la dimensione del blocco di voli puntato da `arrayVoli` non e' piu' fissa (sempre uguale a `MAXVOLI`) durante l'esecuzione del programma, diventa necessario averla memorizzata in una variabile. Questa variabile deve contenere, in qualsiasi momento, il numero di voli effettivamente allocati nel blocco `arrayVoli`, indipendentemente se occupati o disponibili. Quindi deve essere un valore associato stabilmente alla singola tabella (`tab`) con cui si sta lavorando. Ad esempio, durante l'operazione di aggiunta di un nuovo volo si deve verificare che il numero di voli effettivamente memorizzato (`.quantiVoli`) non sia diventato uguale al numero di strutture allocate nel blocco (altrimenti l'aggiunta non si puo' fare subito). Ne concludiamo che la variabile contenente il numero di voli allocati nel blocco `arrayVoli` deve essere un campo della tabella. E sara' il terzo campo, che in `VOLI6.C` abbiamo chiamato `.voliAllocati`.