

Laurea in Ingegneria dell'Informazione

Esercitazioni Guidate di Tecniche della Programmazione

Note introduttive:

- 1) Nel titolo di ogni sezione di questo documento è specificato tra parentesi il nome del (o dei) file in cui è proposta una soluzione (se disponibile nella directory “programmi” di questa EG).
- 2) I programmi che scriveremo dovranno essere in accordo con la definizione standard **ANSI C** del linguaggio C.
 - a. Se usate un sistema diverso dal DEV, provvedete a che la compilazione avvenga con il compilatore standard C.
 - b. Ricordate che un programma C e’ in un file con estensione “.c”
 - c. Se usate il DEVC++, per configurare bene la compilazione bisogna
 - i. andare nel menù “Tools”, selezionare “Compiler Options”, scegliere “Settings” e poi “C Compiler”; poi selezionare almeno “Support all ANSI Standard C Programs”
 - ii. (se l’interfaccia è in italiano ...) andare nel menu’ “Strumenti”, selezionare “Opzioni di compilazione”, “Compilatore”, “Generazione di Codice ...”, “Compilatore C” e poi far apparire “Yes” almeno accanto a “Supporto programmi ANSI standard C.

NB Spesso ci sono svariati esercizi proposti nelle slide delle lezioni; questi esercizi potrebbero essere affrontati durante la EG, anche se non vi compaiono. A meno che non li abbiate già fatti Se li avete affrontati potete sempre farmeli vedere per chiarire eventuali dubbi. Se non li avete affrontati provate a farli ...

8. Esercitazione Guidata 8 – TABELLE e Tabella dei VOLI

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

8.1. Cos’è 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 è l’elenco telefonico (una collezione di elementi; ciascun elemento è l’insieme di dati relativi ad una persona, come il nome, l’indirizzo, il numero di telefono, titoli professionali etc...).

Un altro esempio è l’elenco degli iscritti ad una associazione (anche qui ogni elemento in tabella è una collezione di dati relativi ad una persona, come il nome e indirizzo, ma anche l’anzianità 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 più brevemente, *chiavi*, e vengono usati quando si stanno effettuando operazioni di ricerca e selezione di un certo elemento nella tabella.

8.2. 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 è caratterizzato da

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

La tabella dei voli di quel giorno è una collezione di voli (informazioni su voli) su cui è 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 (cioè 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 è la “chiave di ricerca” per i voli memorizzati.

Scrivere un abbozzo del programma che dovrà 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 menù 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 è in **VOLI1.C**.

Suggerimenti seguono.

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

Usiamo tabelle di 10 voli, per scrivere e testare il programma.

Dopo potremo cambiare questa dimensione

```
#define MAXVOLI 10
```

Un volo (un singolo oggetto che rappresenti un volo) è fatto da una collezione di dati di tipi diversi, quindi la struct è lo strumento da usare qui.

???Come è definita una struct **struct volo {**
Pensarci e poi proseguire.

il tipo “**struct volo**”, cioè il tipo degli oggetti che rappresentano voli aerei nel programma, contiene un campo **oraPartenza**.

L'ora è ben rappresentabile con un'altra struct

Definire **struct ora {**

E poi proseguire

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

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

Sistemato questo, segue il suggerimento sull'uso di **typedef** ... Prova ad usare **typedef** in modo sensato, e poi passa alla prossima pagina

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 limiterà agli elementi
di indice da 0 a **quantivoli-1**.

Quindi una tabella di voli è una struttura a due campi.

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

Segue suggerimento sulle funzioni da definire per costruire il programma richiesto.

Prima, riprendere l'abbozzo di **main()** di questo programma (yes, a volte ritornano): se non avevi inserito le chiamate di funzione, *now would be a good moment* per farlo.

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 sarà 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).

Saranno modificati sia **(*t).arrayVoli** (nell'elemento di indice **quantivoli**) che **(*t).quantivoli**.

L'elemento in cui si inserisce il nuovo volo è
`(*t).arrayVoli[kkk]`

dove **kkk** deve essere uguale al campo **quantivoli** della tabella
`(kkk =(*t).quantivoli)`.

Poi **(*t).quantivoli** deve crescere di uno, perché 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 è 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:
                /* la funzione stampaQuelVolo è rimanadata al prossimo
                   esercizio: qui limitiamo il programma, in modo da
                   sperimentare le funzioni di aggiunta elementi e stampa
                   della tabella
                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 sbagliata\n");
        } /* fine switch */
    } while (scelta!=0);           /* fine do_while*/
}

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

```

8.3. Gestione voli (secondo passo VOLI2.C)

Incrementare il programma costruito al passo precedente, con la funzionalità di stampa di un volo, dato il codice.

Yes, scrivere la funzione che, ricevendo una tabella di voli come definita prima, e un codice, produce la stampa in output delle informazioni sul volo che ha quel codice, se è in tabella.

(E poi testare ...)

Come che cosa? ... testare la funzione scritta ...

Suggerimento 1di2:

La funzione da implementare è la seguente:
... **stampaQuelVolo(...);**

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

Suggerimento 2di2:

Però 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;
cioè 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 */
```

*Realizzare le ultime funzioni discusse sopra e poi
testare il programma – una soluzione è in VOLI2.c*

8.4. 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 ...).

OK, bisogna fare la funzione **eliminavolo** che

- Riceve una tabella e il codice di un volo (quello da eliminare)
- Modifica la tabella, eliminando il volo indicato dal codice dato

Qui bisogna discutere su come fare: le slide della lezione, nella parte finale, fanno vedere un modo per farlo:

Individuare dove è il volo da eliminare (ricerca del volo in base al codice ... è una funzione che abbiamo fatto prima ... restituisce l'indice che il volo cercato ha **nell'arrayVoli** della tabella (oppure -1)

- 1) Se il volo c'è (cioè l'indice restituito al passo 1 è diverso da -1 (**k!=-1**) allora
 - a. Si elimina la stringa destinazione del volo di indice **k** nell'**arrayVoli** della tabella (**free** ...)
 - b. Si copia il contenuto dell'ultimo volo in tabella nel volo di indice **k**
 - c. Si diminuisce di 1 il numero di voli presenti in tabella

Provare ad estendere il programma con questa funzione

Suggerimenti seguono se servono

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(..., char cod[]);
```

```

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 poi si ricopre l'elemento della tabella che ha indice k
copiandoci sopra l'ultimo elemento della tabella;
    t->arrayVoli[k] = t->arrayVoli[ultimo];

(quanto vale l'indice "ultimo"?
Scrivilo qui prima di proseguire ...)

```

INFINE, dopo aver ricoperto il volo da eliminare con l'ultima volo della tabella, bisogna capire che ora la tabella ha un elemento in meno, rispetto a prima, quindi ... che si fa?

Farlo (scrivere l'istruzione) e poi guardare all'inizio della pagina successiva

Si decrementa il campo **quantivoli** della tabella
ad esempio **t->quantivoli-=1;**
*/

Suggerimento 2di2:

Nel caso serva ...

ultimo=t->quantivoli-1;

8.5. Gestione voli (quarto passo VOLI4.C)

Incrementare il programma costruito al passo precedente, con le funzionalità di modifica richieste, cioè

- Funzionalità di cambio dell'ora di partenza di un volo di codice dato
- Funzionalità di 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: Prova a scrivere le chiamate delle funzioni suddette, così come dovrebbero apparire nella `main()` del programma che stai espandendo

Suggerimento successivo ... le funzioni da realizzare potrebbero avere le seguenti intestazioni:

```
int cambiaOraPartenza (TipoTabella *t, ..., int nuovaOra, int nuoviMin);  
  
int cambiaPostiLiberi(..., ..., int k);  
  
...
```

Il parametro tabella qui deve essere l'indirizzo della tabella “attuale” da modificare ... vedi lezione, in corrispondenza del passaggio di un parametro **struct** ...

```
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 */
```

8.6. Gestione voli (quinto passo VOLI5.C)

Incrementare il programma costruito al passo precedente, con le funzionalita` 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 (cioè *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`.

8.7. Gestione voli (sesto passo VOLI6.C)

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

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

- il tipo **TypoTabella** viene modificato, in modo che il campo **arrayVolo** non sia più un array statico di **MAXVOLI** voli, ma il puntatore ad un array di voli dinamicamente allocato: **TypoVolo** * **arrayVoli**;

- **L'inizializzazione** della tabella non si limita più 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** è 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 deallocated il blocco eventualmente puntato da **arrayVoli** e
 - o allocato (e assegnato ad **arrayVoli**) un blocco di voli che verrà 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, così dopo il caricamento dei dati, ci sarà 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 è possibile per mancanza di spazio in **arrayVoli**, anziché lasciar perdere
 - o si fa in modo che **arrayVoli** punti ad un blocco di voli “esteso” (un po’ più 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 già presenti in tabella più qualche altro spazio per nuovi voli, si possono usare varie soluzioni. La più 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 dei suggerimenti elencati sopra.

Insomma salvate il lavoro fatto finora in un file, duplicate questo file cambiandogli nome, e lavorate alle aggiunte al programma su questo secondo file, così ~~se e combinate disastri ... se succedono incidenti ... avete una copia di riserva del lavoro fatto e controllato.~~

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

La scoperta di questa modifica è 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 è più fissa (cioè non è assegnata ad una costante **MAXVOLI**, ma può variare durante l'esecuzione del programma) diventa necessario averla memorizzata da qualche parte ... in una locazione di memoria.

Questa locazione deve contenere, in qualsiasi momento, il numero di voli effettivamente allocati nel blocco **arrayVoli**, indipendentemente se occupati o disponibili.

E questa informazione riguarda la specifica tabella su cui si opera (la specifica variabile **tab**). Quindi deve essere un dato contenuto nella variabile **tab** ... uno dei suoi campi!

Ad esempio, potremmo chiamare questo TERZO CAMPO della variabile **tab**, cioe` terzo campo del tipo **TipоТабелла**, col nome **voliAllocati**.

Allora, durante l'operazione di aggiunta di un nuovo volo in **tab**, bisogna verificare che che il numero di voli effettivamente memorizzato (**.quantivoli**) non sia diventato uguale al numero di spazi-volo allocati nel blocco **arrayVoli** (che non è più una costante ma è **voliAllocati**):

? tab.quantivoli < tab.voliAllocati ?

Se non è così, l'aggiunta non si può fare subito e bisogna aspettare che qualche volo venga cancellato.

In **VOLI6.C** abbiamo applicato questo ragionamento.