Fondamenti di Informatica e Laboratorio T-AB
Ingegneria Elettronica e Telecomunicazioni

Lab 09
Strutture e programmi su più file

Alcune Informazioni Utili

Funzioni per manipolare stringhe (in string.h)

- Copia di stringhe (“src” in “dst”):
  `strcpy(char *dst, const char *src);`
- Concatenazione (risultato in “s1”):
  `strcat(char *s1, const char *s2);`
- Confronto (-1 se “s1” viene prima di “s2”, 0 se sono uguali e +1 se “s1” viene dopo “s2”):
  `strcat(char *s1, const char *s2);`

Include Guards

Per evitare definizioni multiple di una struttura quando si lavora con più file, si utilizza una “include guard”. Es:

Nel file “funzioni.h”:

```
#ifndef FUNZIONI_H
#define FUNZIONI_H

typedef struct {
          int real; int imm; } Complex;

#endif
```

Esercizio 1

- Sia data la struttura

```
struct time
{
  int hour, minute, second;
};
```

- Per semplicità si può definire il tipo Time

```
typedef struct time Time;
```

Esercizio 1

Si realizzi in un modulo **tempo.h/tempo.c** un insieme di funzioni per la gestione del tipo Time. In particolare:

- Si realizzi una funzione

```
Time leggiTime()
```

che legga da input ore, minuti e secondi, e restituisca una struttura di tipo Time opportunamente inizializzato coi valori letti

- Si realizzi una funzione

```
int leggiMoreTimes(Time v[], int dim)
```

La funzione deve leggere da input delle strutture Time (a tal scopo si utilizzi la funzione definita sopra) e salvarle nel vettore v, di dimensione fisica dim. La funzione deve restituire il numero di elementi letti. La lettura termina se l'utente inserisce un tempo con ora negativa.

Esercizio 1

- Si progetti una funzione in grado di calcolare la differenza fra due strutture **`Time`** e che restituisca il risultato in termini di una nuova struttura **`Time`**

- L'interfaccia della funzione è facilmente desumibile dalle specifiche:

```
Time subtract(Time t1, Time t2);
```

- Due possibili approcci:
  1. Trasformare in secondi, eseguire la differenza, trasformare in ore, minuti, secondi
  2. Eseguire la sottrazione direttamente tenendo conto dei riporti

Esercizio 2

Una compagnia di autobus che effettua servizio su lunghe distanze vuole realizzare un programma di controllo delle prenotazioni dei posti.

A tal scopo rappresenta ogni prenotazione tramite una struttura **`booking`** contenente nome del cliente (al massimo 1023 caratteri, senza spazi) e numero del posto prenotato (un intero).

Le prenotazioni effettuate vengono registrate tramite un array (di dimensione prefissata **`DIM`**) di strutture **`booking`**, di dimensione logica iniziale pari a 0.

Si realizzi il modulo C gestione.h/gestione.c, contenente la struttura dati booking e le seguenti funzioni…

Esercizio 2

a) Si realizzi una funzione:

```
int leggi(booking * dest);
```

La funzione legge da input una struttura di tipo booking, e provvede a memorizzarla in dest. La funzione deve restituire 1 se è stata letta una nuova prenotazione, 0 altrimenti (cioè nel caso in cui il nome del passeggero è “fine”).

Esercizio 2

b) Si realizzi una funzione:

```
int assegna( booking list[],
             int dim,
             int * lengthList,
             char * name,
             int pref)
```

La funzione riceve in ingresso l'array di prenotazioni e la sua dimensione fisica e logica, e poi il nome del cliente ed il posto da lui indicato. La funzione deve controllare che il posto indicato non sia già stato assegnato, ed in caso contrario deve restituire il valore 1.

Esercizio 2

Qualora invece il posto sia ancora libero, la funzione deve assegnare tale posto al cliente copiando i dati della prenotazione nell'ultima posizione libera nell'array, e deve provvedere ad aggiornare correttamente la dimensione logica dell'array. In questo secondo caso la funzione deve invece restituire come valore uno 0, indicante il successo nella prenotazione.

Al fine di copiare il nome del cliente, si utilizzi la funzione di libreria

```
char * strcpy(char * s, char * ct)
```

che copia ct in s (terminatore compreso).

Esercizio 2

c) Si realizzi un programma main (file main.c) che chieda all'operatore il nome di un utente, e di seguito il posto prescelto (a tal fine si usi la funzione di cui al punto a)). Il programma deve cercare di registrare la prenotazione tramite la funzione `assegna`; qualora l'operazione di prenotazione fallisca (perché il posto risulta essere già assegnato), il programma provveda a chiedere all'operatore un nuovo posto, finché non si riesca ad effettuare la prenotazione.

Esercizio 2

Qualora l'operatore inserisca il nome "fine", il programma deve terminare; qualora invece venga inserita la stringa "stampa", il programma deve stampare a video le prenotazioni già effettuate.

A tal scopo si usi la funzione di libreria:

```
int strcmp(char * ct, char * cs)
```

che restituisce 0 se e solo se le due stringhe sono identiche (lessicograficamente).

Esercizio 3

Si realizzi il modulo `voli.h/voli.c` per la gestione di viaggi in aereo.

Per memorizzare le informazioni su una singola tratta, si definisca la struttura “Flight”, contenente il nome della città di partenza e della destinazione (stringhe senza spazi di al più 50 caratteri), insieme all’orario di partenza e di arrivo, memorizzate come strutture Time (quelle realizzate nell’esercizio 1).

Nota: per questo esercizio si richiede di sfruttare il modulo precedentemente sviluppato, che va mantenuto nella coppia di file “tempo.h/tempo.c”

Esercizio 3

a) Si realizzi una funzione:

```
int readFlights(Flight * f,
                int* n, int nmax);
```

La funzione deve leggere da input le informazioni relative ad uno o più voli e memorizzarle nel vettore “f”. La lettura termina quando l’utente inserisce come città di partenza “fine”. Al termine della lettura, la variabili intere “n” deve contenere il numero di voli letti. In nessun caso devono essere letti più di “nmax” voli.

Si realizzi un semplice main di prova.

Esercizio 3

a) Si realizzi una funzione:

```
int findFlights(Flight* f, int n,
                char* departure,
                char* destination);
```

La funzione deve stampare a video le informazioni su tutti i voli dalla città “departure” alla città “destination”, presenti nell’array “flights”, di dimensione logica “n”.

Si realizzi un semplice main di prova, estendendo quello del passo precedente.

Esercizio 3

a) Si realizzi una funzione:

```
int findHops(Flight* f, int n,
             char* departure,
             char* destination);
```

La funzione deve stampare a video le informazioni su tutte le coincidenze da “departure” a “destination”. Una coincidenza è una coppia di voli in cui la destinazione del primo coincide con la partenza del secondo e l’orario di arrivo del primo precede quello di partenza del secondo di almeno 2 ore.

Si realizzi un semplice main di prova, estendendo quello del passo precedente.

Esercizio 4

Si realizzino i moduli `oggetti.h/oggetti.c`, `magazzino.h/magazzino.c` e `vendite.h/vendite.c` per la gestione di una attività commerciale.

Il modulo `oggetti.h/oggetti.c` deve definire la struttura “Articolo” (definita con una typedef), che rappresenta uno degli articoli trattati. Ogni articolo ha un nome (al più 40 caratteri), un costo ed un prezzo (numeri reali).

Esercizio 4

Inoltre, il modulo `oggetti.h/oggetti.c` deve definire la procedura:

```
void leggiListino(int maxdim, int* dim,
                  Articolo listino[]);
```

Che richiede all'utente di inserire le informazioni relative agli articoli trattati. La procedura utilizzerà tali informazioni per riempire l'array "listino", la cui dimensione logica sarà disponibile al termine dell'esecuzione nell'intero "dim". La procedure termina quando l'utente inserisce un articolo con nome "fine", o quando "maxdim" articoli sono stati inseriti.

Esercizio 4

Il modulo `magazzino.h/magazzino.c` deve definire la procedura:

```
void leggiInventario(int dim,Articolo listino[],
                     int inventario[]);
```

Che richiede all'utente di inserire le quantità disponibili per ciascuno degli articoli in listino. La procedura deve stampare il nome di ogni articolo prima di richiederne la quantità disponibile (che deve essere non-negativa). La procedura memorizza tali quantità nell'array "inventario". L'intero "dim" rappresenta la dimensione del listino.

Esercizio 4

Il modulo `magazzino.h/magazzino.c` deve definire le funzione:

```
int preleva(int i, int dim, int inventario[]);
```

Che riduce di una unità la quantità dell'articolo di indice "i" (se l'articolo è ancora disponibile). Si controlli che tale indice sia >= 0 e minore della dimensione del listino "dim". La funzione restituisce 1 in caso l'operazione sia effettuata correttamente e 0 in caso contrario.

Esercizio 4

Il modulo `vendite.h/vendite.c` deve definire le procedura:

```
void vendi(int i, int dim,
           Articolo listino[], float* capitale);
```

Che incrementa il valore di “capitale” di una quantità pari al profitto (prezzo – costo) dell’articolo di indice “i” in “listino”. L’intero “dim” rappresenta la dimensione del listino.

Sia realizzi poi un main che richiede all’utente i dati sul listino, sulle quantità di articoli disponibili e proceda quindi alla vendita di un articolo utilizzando sia “vendi” che “preleva”.