



LE STRUTTURE

Manuale linguaggio C

Le strutture

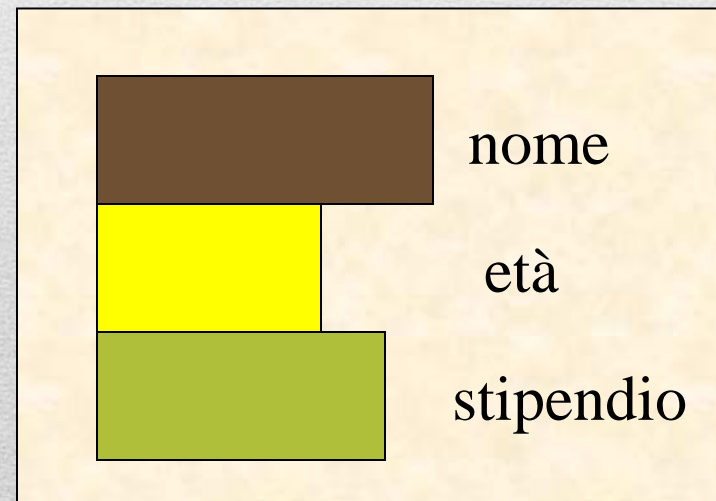
Una struttura è una collezione finita di variabili non necessariamente dello stesso tipo, ognuna identificata da un nome. Definizione di una **variabile** di tipo struttura:

```
struct [<etichetta>] {  
    <definizione-di-variabile>  
} <nome-struttura>;
```

Esempio:

```
struct persona {  
    char nome[20];  
    int eta;  
    float stipendio;  
}pers ;
```

Struct persona



Esempio

Supponiamo di voler memorizzare, relativamente a ciascun abitante di un dato comune, nome e cognome, data di nascita e codice fiscale:

- Nome e cognome costituiscono un array di caratteri
- La data è composta da tre numeri interi, che descrivono giorno, mese ed anno
- Il codice fiscale è un array di 16 caratteri (15 per il codice ed uno per il carattere nullo di terminazione)

Le informazioni non possono essere collezionate in un unico array di caratteri, perché sono disomogenee

Esempio

Possibile soluzione: memorizzare le informazioni in variabili distinte (perdo la possibilità di accedervi in maniera unitaria)

```
char name[40], fiscode[16];  
short day, month, year;
```

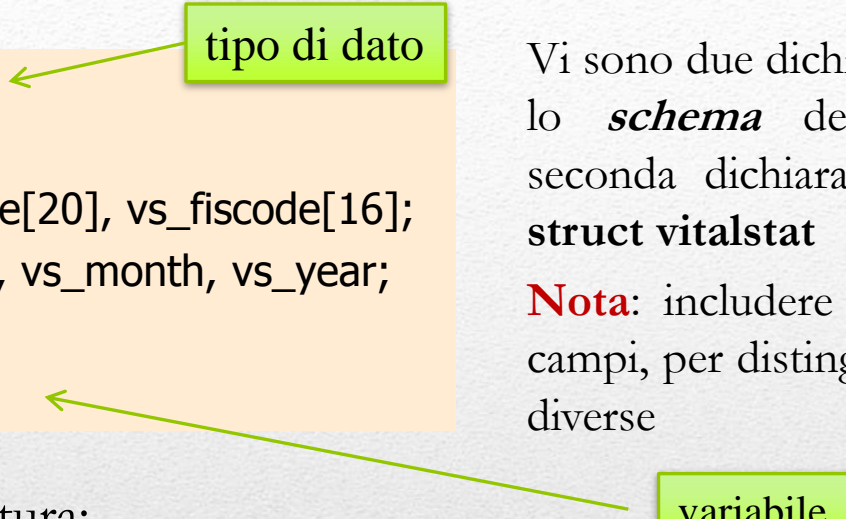
Le informazioni relative ad una persona sono sparse per la memoria e non è possibile accedervi in maniera unitaria

Un'organizzazione più naturale consiste nella definizione di una variabile che contenga tutti i dati relativi ad una persona, utilizzando un tipo **struttura**

Una struttura è simile ad un array, ma è costituita da **campi** (piuttosto che da elementi) che sono identificati da **nomi** (piuttosto che da indici) e possono contenere informazione di tipo diverso

Dichiarazione delle struct

```
struct vitalstat
{
    char vs_name[20], vs_fiscode[16];
    short vs_day, vs_month, vs_year;
};
struct vitalstat vs;
```



Vi sono due dichiarazioni: la prima contiene lo ***schema*** della struttura *vitalstat*, la seconda dichiara una variabile *vs* di tipo **struct vitalstat**

Nota: includere un prefisso nei nomi dei campi, per distinguerli da campi di strutture diverse

I campi di una struttura:


- devono avere nomi univoci all'interno di una struttura
- strutture diverse possono avere campi con lo stesso nome
- i nomi dei campi possono coincidere con nomi di variabili o funzioni

La variabile che è stata creata a partire da una struttura, rappresenta tutta la zona di memoria occupata dalla struttura stessa (in modo analogo alle variabili di tipo primitivo) e non solo il riferimento al suo inizio. Questa distinzione è importante, per non fare confusione con il comportamento relativo agli array, che in realtà sono solo dei puntatori.

Dichiarazione delle struct

```
struct vitalstat  
{  
    char vs_name[20], vs_fiscode[16];  
    short vs_day, vs_month, vs_year;  
} vs;
```

tipo di dato e variabile




```
struct vitalstat  
{  
    char vs_name[20], vs_fiscode[16];  
    short vs_day, vs_month, vs_year;  
} vs, vs1, vs2;
```

tipo di dato e variabili



```
struct  
{  
    char vs_name[20], vs_fiscode[16];  
    short vs_day, vs_month, vs_year;  
} vs;
```

tipo di dato anonimo e variabile



Esempio

Il nome *vitalstat* è un'**etichetta** (o tag) e *struct vitalstat* rappresenta un nuovo tipo di dati definito dall'utente, per il quale non viene riservata memoria

L'etichetta può essere utilizzata ogni volta che è necessario creare ulteriori variabili che contengano gli stessi campi

```
struct vitalstat vsa[1000], *pvs;  
pvs = &vsa[10];
```

La sintassi della dichiarazione di una struttura può assumere diverse forme:

- Dichiarazione dell'etichetta e uso dell'etichetta (insieme alla parola chiave *struct*) per definire le variabili
 - Uso di *typedef* per definire un particolare tipo struttura
-

Esempio

sinonimo di tipo



Il tipo VITALSTAT rappresenta l'intera struttura, compresa la parola chiave **struct**

- Il nome di tipo è scritto maiuscolo, per distinguerlo dai nomi di etichette e variabili
- Un'etichetta o un **typedef** consentono di definire una sola volta lo schema di una struttura, per dichiarare variabili del tipo struttura quando necessario

```
typedef
struct
{
    char vs_name[20], vs_fiscode[16];
    short vs_day, vs_month, vs_year;
}
VITALSTAT;
VITALSTAT vs;
```

Le dichiarazioni di struttura sono normalmente raggruppate in **file header**, così da poter essere accedute da più file sorgente

Inizializzazione delle strutture

Una struttura può essere inizializzata facendo seguire al nome della variabile di tipo struttura il simbolo di uguale e la lista dei valori iniziali racchiusi tra parentesi graffe

Ogni valore iniziale deve essere dello stesso tipo del corrispondente campo della struttura

```
VITALSTAT vs = {"Marco Rossi", "MRCRSS89C23D612K",  
                23, 3, 1989 };
```

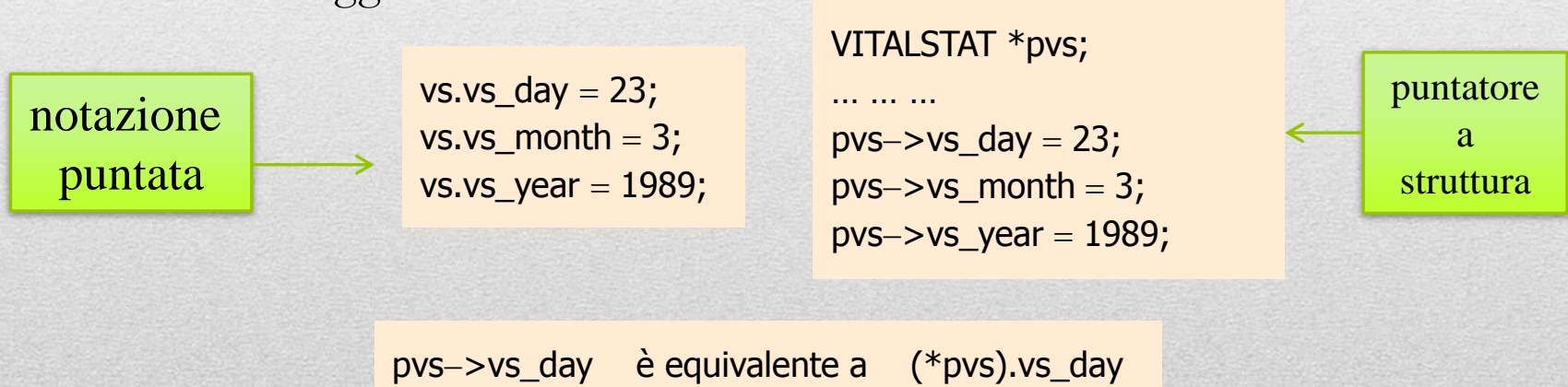
Un valore iniziale non può essere incluso in una dichiarazione che contiene solo un'etichetta o un **typedef**, poiché tali dichiarazioni definiscono lo schema della struttura, ma non riservano memoria

```
typedef struct  
{  
    int a;  
    float b;  
} s = {1, 1.0}; //SBAGLIATO
```

L'accesso ai campi delle strutture

Esistono due modalità diverse di accesso ai campi di una struttura: **diretta** o **mediante puntatore**

- Nel caso di accesso **diretto** alla struttura, si usano il nome della struttura ed il nome del campo, separati dall'operatore punto “.” Ogni campo si comporta e si usa come una normale variabile. Non c'è alcuna ambiguità perché ogni variabile è definita nel proprio environment.
- Nel caso di accesso **tramite puntatore** alla struttura, si usa l'operatore freccia destra “->”, formato dalla concatenazione del segno meno e del simbolo maggiore



L'operatore -> è una forma abbreviata per accedere al contenuto puntato dal puntatore e quindi applicare l'operatore punto

Le strutture annidate

Una **struttura annidata** è una struttura in cui almeno uno dei campi è, a sua volta, una struttura

Le strutture annidate sono comuni in C perché consentono di creare gerarchie di dati

```
typedef struct
{
    char vs_name[20], vs_fiscode[16];
    struct vs_birth_date
    {
        short vs_day;
        short vs_month;
        short vs_year;
    };
} VITALSTAT;
```

Per identificare l'anno di nascita in una struttura *vs* dichiarata VITALSTAT, si deve scrivere:

`vs.vs_birth_date.vs_year`

```
typedef struct
{
    char day;
    char month;
    short year;
} DATE;

typedef struct
{
    char vs_name[20], vs_fiscode[16];
    DATE vs_birth_date;
} VITALSTAT;
```

Le strutture annidate

Una struttura annidata viene inizializzata racchiudendo i valori iniziali fra parentesi graffe

```
typedef struct
{
    DATE d;
    char event[20];
} CALENDAR;

CALENDAR holiday = {{25, 12, 2013}, "Natale"};
```

Non esistono limiti al numero di annidamenti delle strutture, anche se i riferimenti agli elementi diventano sempre più difficili da leggere perché contengono i nomi di tutte le strutture intermedie

Assegnamento di strutture

A differenza di quanto accade con gli array, il nome della struttura rappresenta la struttura nel suo complesso. Quindi, è possibile:

- assegnare una struttura a un'altra
- che una funzione restituisca una struttura
- passare una struttura come parametro a una funzione, cioè passarne una copia

```
struct {  
    int a;  
    float b;  
} s1, s2, *ps;
```

... ..

```
s1 = s2;  
s2 = sf();  
ps = &s1;  
s2 = *ps;
```

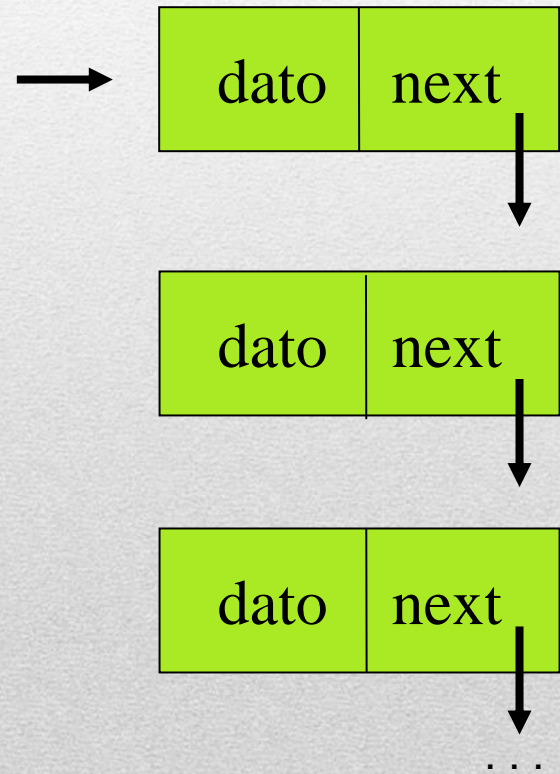
... ..

Puntatori all'interno di strutture

Vogliamo definire una struttura dati che serva come nodo di una lista semplice, una struttura che contenga cioè, ad esempio, un intero (il dato della lista) ed un puntatore all'elemento successivo della lista.

Come definire un puntatore ad un tipo di dato mentre ancora il tipo di dato non è stato definito ?

```
struct nodolista {  
    int id;  
    struct nodolista *next;  
};  
struct nodolista nodo;  
/* oppure: */  
typedef struct nodolista {  
    int id;  
    struct nodolista *next;  
} NODOLISTA;  
NODOLISTA nodo;
```



Puntatori all'interno di strutture

Il **riferimento in avanti** a strutture è uno dei pochi casi nel linguaggio C in cui un identificatore può essere utilizzato prima di essere dichiarato

Il riferimento in avanti non è ammesso con la definizione di tipo (**typedef**)

```
typedef struct nodolista {  
    int id;  
    NODOLISTA *next;      /* errato perché NODOLISTA  
                           * non è ancora stato dichiarato  
                           */  
} NODOLISTA;  
NODOLISTA nodo;
```

Array all'interno di strutture

Se una struttura, anche molto voluminosa, viene copiata elemento per elemento...

.. perché non usare una struttura per incapsulare un array?

In effetti il C non rifiuta di manipolare gli array nella loro interezza “per principio”: è solo la conseguenza del modo in cui si interpreta il loro nome

```
main() {  
    struct string20 {  
        char s[20];  
    } s1 = {"Paolino Paperino" }, s2 = {"Gastone Fortunato" };  
    s1 = s2;      /* FUNZIONA!! */  
}
```

Usando una struttura per “racchiudere” un array, si fornisce all’array esattamente quello che gli mancava: un modo per denotare “il tutto”, ossia, un “contenitore” dotato di nome, che consente di riferirsi all’array nella sua globalità.

Array di strutture

Non ci sono vincoli riguardo al tipo degli elementi di un vettore: essi possono anche essere struct. Ad esempio:

```
typedef struct {  
    char Nome[20];  
    char Cognome[20];  
    int Reddito;  
    float Aliquota;  
} contribuente;  
contribuente archivio[1000];
```

archivio è un vettore di 1000 elementi, ciascuno dei quali è di tipo contribuente

Abbiamo realizzato un vettore di record, o struttura tabellare.

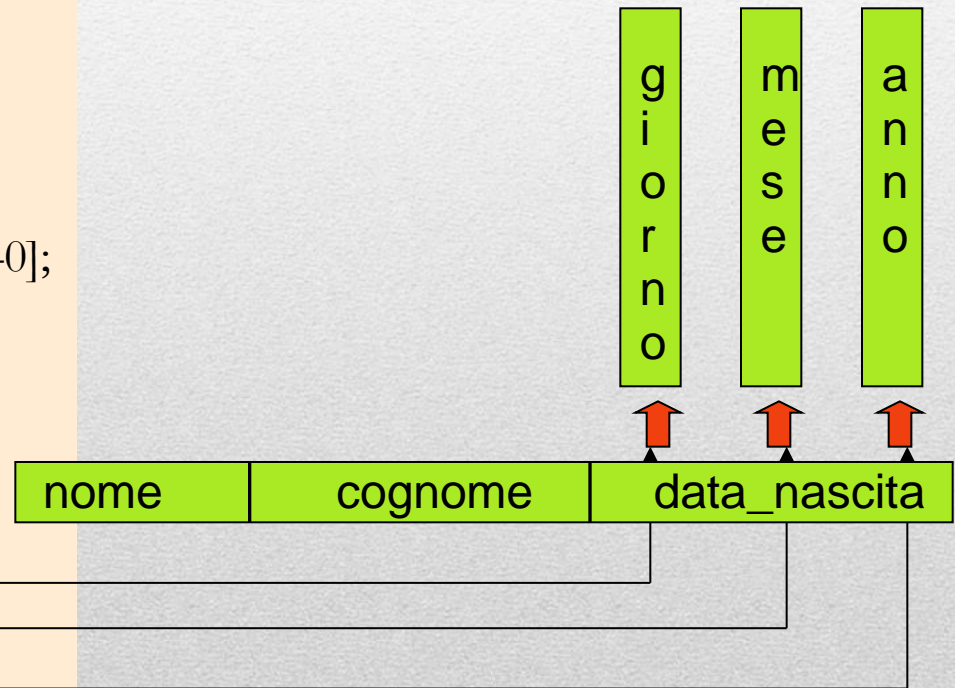
Vettori e strutture

Allo stesso modo, si possono fare record di vettori e record di record. Ad esempio:

```
typedef struct{
    int giorno;
    int mese;
    int anno;
} data;

typedef struct{
    char nome[20];
    char cognome[40];
    data data_nasc;
} persona;

persona P;
...
P.data_nasc.giorno=25;
P.data_nasc.mese=3;
P.data_nasc.anno=1992;
```



Esercizio

Scrivere un programma che acquisisca i dati relativi agli studenti di una classe formata al massimo da 20 alunni:

- nome
- cognome
- voti: rappresenta i voti dello studente in 3 materie (italiano, matematica, inglese);

Il programma deve successivamente calcolare e stampare, per ogni studente, la media dei voti ottenuti nelle 3 materie. Introduciamo un tipo di dato per rappresentare il generico studente:

```
typedef struct {  
    char nome[30];  
    char cognome[30];  
    int voto[3];  
} studente;
```

La classe è rappresentata da un vettore di studenti:

```
studente classe[20];
```

Esercizio

```
#include <stdio.h>
#define ita 0
#define mat 1
#define ing 2
#define N 20

typedef struct{
    char nome[30],
    cognome[30];
    int voto[3];
} studente;

main()
{
    studente classe[N];
    float m;
    int i; int j;
```


Esercizio

```
/* lettura dati */
for(i=0;i<N; i++)
{
    gets(classe[i].nome);
    gets(classe[i].cognome);
    for(j=ita; j<=ing; j++)
        scanf( "%d", &classe[i].voto[j]);
}

/* stampa delle medie */
for( i=0; i<N; i++ )
{
    for( m=0,j=ita; j<=ing; j++ )
        m += classe[i].voto[j];
    printf( "media di %s %s: %f\n", classe[i].nome, classe[i].cognome, m/3 );
}
}
```

Esercizio

Utilizzando un array di strutture, si vogliono memorizzare i seguenti dati relativi agli **studenti**:

- Nome,
- Cognome,
- Matricola,
- Anno di corso,
- Elenco dei voti (memorizzato in un vettore).

Si utilizza, poi, un ulteriore vettore di strutture per memorizzare le informazioni relative alle **materie**:

- Codice materia,
- Descrizione,
- Docente.

L'indice con cui sono memorizzati i voti all'interno dell'array di voti corrisponde alla posizione in cui sono memorizzate le materie nel rispettivo vettore delle materie.

Sia data la matricola di uno studente. Calcolare la sua media e visualizzare le materie in cui risulta insufficiente, il docente della materia e il voto corrispondente.

Esercizio

.c | unireal.cpp | esercizio_1.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

#define MAX_STUD 500
#define N_MAT 5
#define VOTOMIN 1
#define VOTOMAX 11
#define SUFF 6
#define TRUE 1
#define FALSE 0

// Studenti
struct
{
    char cognome[50];
    char nome[50];
    int matricola;
    int anno_di_corso;
    float voto[N_MAT]; //elenco dei voti dello studente nelle diverse materie
}stud[MAX_STUD];

// Materie
struct
{
    int codice;
    char descr[50];
    char docente[50];
} materia[N_MAT] = {{1,"Lettere","Rossi"},{2,"Matematica","Monti"},{3,"Inglese","Smith"},{4,"Diritto","Bianchi"},{5,"Fisica","Verdi"}};
```

Esercizio

```
int main()
{

    float media, somma;
    char next, trovato;
    int i, j, nStud, matricola, num_insuff;

    // inizializzazione dimensioni
    do
    {
        printf("Quanti studenti (max %d)? ", MAX_STUD);
        scanf("%d", &nStud);
    } while (nStud > MAX_STUD || nStud <= 0);

    // inserimento studenti e voti
    printf("\n\nInserimento STUDENTI e VOTI...\n");

    for (i = 0; i < nStud; i++)
    {
        printf("\nStudente[%d]:\n", i);
        printf("Matricola: ");
        scanf("%d", &stud[i].matricola);
        printf("Nome: ");
        scanf("%s", stud[i].nome);
        printf("Cognome: ");
        scanf("%s", stud[i].cognome);
        printf(" Voti studente:\n", i);

        for (j = 0; j < N_MAT; j++)
        {
```


Esercizio

```
for(j = 0; j < N_MAT; j++)
{

    printf("\t%s: ",materia[j].descr);
    do
    {
        printf("Inserisci voto studente compreso fra MIN e MAX\n");
        scanf("%f",&stud[i].voto[j]); //inserisce i voti per ciascuna materia nel vettore dei voti dello studente i-esimo
    }while(stud[i].voto[j] > VOTOMAX || stud[i].voto[j] < VOTOMIN);
    |

}

}
```

Esercizio

```
// Inserimento matricola da cercare
do
{
    trovato = FALSE;
    do
    {
        printf("\n\nInserisci la matricola dello studente da ricercare: ");
        scanf("%d",&matricola);

        for (i = 0; i < nStud && !trovato; i++)
        {
            if (stud[i].matricola==matricola)
                trovato=TRUE;
        }

        if (!trovato)
            printf("La matricola inserita non esiste.");
    }
    while (!trovato);
}
```


Esercizio

```
// Ricerca insufficienze e calcolo media
somma = 0;
num_insuff = 0;
printf("Materie insufficienti:\n");

i--; // indice della matricola cercata

for(j = 0; j < N_MAT; j++)
{
    somma += stud[i].voto[j];
    if (stud[i].voto[j] < SUFF) // verifica materie insufficienti
    {
        printf("%s (Prof. %s) %.1f\n", materia[j].descr, materia[j].docente, stud[i].voto[j]);
        num_insuff++;
    }
}

if (!num_insuff)
    printf("Nessuna materia insufficiente");

// calcolo media
media = somma/N_MAT;
printf("\nMedia: %.1f\n", media);

do
{
    printf("\n\nVuoi eseguire un'altra ricerca (S/N)?");
    scanf("\n%c", &next);
}while ((toupper(next)) != 'S' && (toupper(next)) != 'N');

}
while ((toupper(next)) == 'S');

return 0;
}
```

Progetto di programmazione – prefisso tel.

Scrivere un programma C che chieda all'utente di immettere un prefisso telefonico internazionale e poi lo cerchi nel vettore *country_codes*. Se il programma trova il prefisso, allora deve visualizzare il nome della nazione corrispondente. Altrimenti, il programma deve stampare un messaggio di errore.

Creiamo una struttura che possa mantenere il nome della nazione insieme al suo prefisso:

```
struct dialing_code {  
    char *country;  
    int code;  
};
```

Il nome è rappresentato tramite un puntatore a carattere; il campo *country* punterà quindi a una stringa letterale. Dichiariamo poi un vettore di queste strutture e lo inizializziamo per contenere i codici di alcune nazioni.

Progetto di programmazione – prefisso tel.

```
const struct dialing_code country_codes[] =  
    {"Argentina",          54}, {"Bangladesh",          880},  
    {"Brazil",             55}, {"Burma (Myanmar)",       95},  
    {"China",              86}, {"Colombia",              57},  
    {"Congo, Dem. Rep. of", 243}, {"Egypt",            20},  
    {"Ethiopia",           251}, {"France",              33},  
    {"Germany",            49}, {"India",                91},  
    {"Indonesia",          62}, {"Iran",                98},  
    {"Italy",              39}, {"Japan",                81},  
    {"Mexico",             52}, {"Nigeria",            234},  
    {"Pakistan",           92}, {"Philippines",         63},  
    {"Poland",             48}, {"Russia",              7},  
    {"South Africa",       27}, {"South Korea",       82},  
    {"Spain",              34}, {"Sudan",             249},  
    {"Thailand",           66}, {"Turkey",           90},  
    {"Ukraine",           380}, {"United Kingdom",      44},  
    {"United States",      1}, {"Vietnam",          84}};
```

Progetto di programmazione – prefisso tel.

```
#include <stdio.h>
```

```
#define COUNTRY_COUNT \  
((int) (sizeof(country_codes) / sizeof(country_codes[0])))
```

```
struct dialing_code {  
    char *country;  
    int code;  
};
```

```
const struct dialing_code country_codes[] = //vedi lucido precedente
```

```
int main(void)  
{  
    int code, i;
```

```
    //CONTINUA
```

Progetto di programmazione – prefisso tel.

```
//CONTINUA
```

```
printf("Enter dialing code: ");  
scanf("%d", &code);
```

```
for (i = 0; i < COUNTRY_COUNT; i++)  
    if (code == country_codes[i].code) {  
        printf("The country with dialing code %d is %s\n",  
            code, country_codes[i].country);  
        return 0;  
    }
```

```
printf("No corresponding country found\n");  
return 0;  
}
```
