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*:

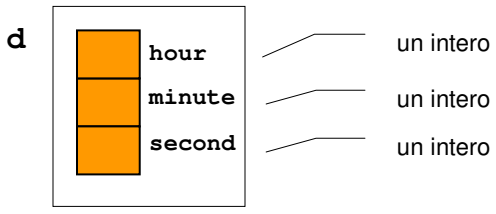
```
struct [<etichetta>]
{
   { <definizione-di-variabile> }
} <nomeStruttura>
```

1

Strutture

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

La variabile `t` è composta da tre interi di nome `hour`, `minute` e `second`



2

Strutture

- **`time`** è solo un'etichetta, è opzionale e serve per dichiarare altre variabili dello stesso tipo
- **`struct time t1, t2;`**
  → dichiara due variabili **`t1`** e **`t2`** di tipo struttura **`time`**
- L'accesso ai campi delle strutture avviene tramite la notazione puntata:

```
t1.hour = 12;
t1.minute = 55;
t1.second = 23;
printf("It's %d:%d:%d; it's time for LUNCH!",
    t1.hour, t1.minute, t1.second);
```

3

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* (copia!)
  - **`f2 = f1;`**
- *che una funzione restituisca una struttura* (restituzione di una copia!)
  - struct time getNoon()
    {
    struct time t;
    t.hour = 12; t.minute = 0; t.second = 0;
    return t;
    }
- *passare una struttura come parametro a una funzione* (passaggio di una copia!)

4

Strutture & Array

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 come un tutt'uno "per principio": è solo la conseguenza del modo in cui si interpreta il loro nome
- quindi, *"chiudendoli in una struttura"* forse ci si riesce

5

Strutture & Array

```
main(){
  struct string20
  {
     char s[20];
  } s1 = {"Paolino Paperino" },
    s2 = {"Gastone Paperone" };

  s1 = s2;    /* FUNZIONA!! */
}
```

- È fondamentale ricordare che si stanno assegnando strutture che contengono array e non array direttamente!!

6

Strutture & Array

Allo stesso modo, una funzione può "restituire" un array:

```
struct string20 { char s[20]; } ;
struct string20 maiusc(struct string20 x)
{
  int k;
  for (k = 0; k < strlen(x.s); k++)
      x.s[k] = toupper(x.s[k]);
  return x;
}

main()
{
 struct string20 m = {"Che bello!"}, mm;
 mm = maiusc(m);
 printf("%s", mm);
}
```

7

Strutture & Array

- Usando una struttura per “racchiudere” un array, *si fornisce all’array esattamente quello che gli mancava:*
  - un modo per denotare “*il tutto*”

    in altre parole
  - un “contenitore” dotato di nome, c*he consenta di riferirsi all’array nella sua globalità*

8

Esercizio

- Sia data la struttura

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

- 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`**

9

Esercizio

- Per semplicità si può definire il tipo Time

```
typedef struct time 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

10

Esercizio

```
Time subtract1(Time t1, Time t2)
{
    int s1, s2, sResult;
    Time result;

    s1 = t1.hour * 3600 + t1.minute * 60 + t1.second;
    s2 = t2.hour * 3600 + t2.minute * 60 + t2.second;
    sResult = s1 - s2;

    result.hour = sResult / 3600;
    sResult = sResult % 3600;
    result.minute = sResult / 60;
    sResult = sResult % 60;
    result.second = sResult;

    return result;
}
```

11

Esercizio

```
Time subtract2(Time t1, Time t2)
{
    Time result;
    int carry;
    result.second = t1.second - t2.second;
    carry = 0;
    if (result.second < 0)
    {
        result.second = 60 + result.second;
        carry = -1;
    }
    result.minute = t1.minute - t2.minute + carry;
    carry = 0;
    if (result.minute < 0)
    {
        result.minute = 60 + result.minute;
        carry = -1;
    }
    result.hour = t1.hour - t2.hour + carry;
    return result;
}
```

12

Strutture innestate

- Ovviamente (?) non ci sono problemi ad innestare strutture in altre strutture
- Ad esempio si può pensare avere una struttura *address* contenuta nella struttura *person*
- Come esercizio si può pensare di fornire alcune funzioni (servizi) che consentano di operare in modo agevole con le strutture di cui sopra
- Per cominciare:
  - Operazioni di lettura da console
  - Operazioni di formattazione su stringa

13

Person & Address – Definizioni

```
#ifndef PERSONTYPEDEFS
#define PERSONTYPEDEFS

typedef struct addressStruct
{
    char street[80];
    char postalCode[8];
    char city[30];
    char state[20];
} Address;
```

```
typedef struct personStruct
{
    char firstName[50];
    char secondName[50];
    char phone[18];
    char cell[18];
    Address address;
} Person;

#define PERSONARRAYDIM 100

typedef Person
   PersonArray[PERSONARRAYDIM];

#endif
```

14

Person & Address – Note

- Uso di

```
#ifndef PERSONTYPEDEFS
#define PERSONTYPEDEFS
```

  per evitare definizioni multiple

- La definizione del tipo Address viene prima della definizione di Person; Person include un Address quindi deve averne visibilità → Le regole sono sempre le stesse!

- Definizione di un array di Person... Non si sa mai!

15

Person & Address – Servizi

- Lettura da console

```
Address readAddressFromConsole();
Person readPersonFromConsole();
```

- Formattazione su stringa

```
void formatAddress(char str[200], Address addr);
void formatPerson(char str[400], Person prs);
```

Tutte le strutture sono passate per valore!!

16

Person & Address – Servizi

```
Person readPersonFromConsole()
{
    Person prs;
    printf("First name: ");
    gets(prs.firstName);
    printf("Second name: ");
    gets(prs.secondName);
    printf("Phone: ");
    gets(prs.phone);
    printf("Cell.: ");
    gets(prs.cell);
    prs.address = readAddressFromConsole();
    return prs;
}
```

Notare che:

1. *È stata incapsulata la logica di lettura di Person e Address in funzioni diverse → le strutture sono "logicamente" indipendenti*
2. ***`readAddress`**... funzionerà in maniera simile*

17

Person & Address – Servizi

```
void formatPerson(char str[400], Person prs)
{
    char addrStr[200];
    formatAddress(addrStr, prs.address);
    sprintf(str, "First name: %s\nSecond name: %s\nPhone: "
        "%s\nCell.: %s\n%s",
        prs.firstName, prs.secondName, prs.phone, prs.cell,
        addrStr);
}
```

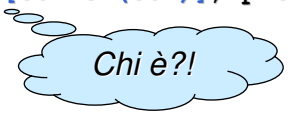
E' proprio necessario creare un buffer aggiuntivo per contenere provvisoriamente l'indirizzo?

18

Person & Address – Servizi

- Effettivamente un modo c'è...

```
void formatPerson(char str[400], Person prs)
{
    sprintf(str, "First name: %s\nSecond name: "
      "%s\nPhone: %s\nCell.: %s\n",
      prs.firstName, prs.secondName, prs.phone,
      prs.cell);
    formatAddress(&str[strlen(str)], prs.address);
}
```



19

Person & Address – Altri servizi

- Si supponga di voler costruire un programma di gestione di una rubrica
- Attualmente esistono i servizi per:
  - Inserire un nuovo contatto
  - Stampare un contatto
- Evidentemente mancano:
  - Ricerca di un contatto (per cognome, anche parziale)
  - Eliminazione di un contatto
  - Memorizzazione in memoria permanente (file?)
- Per mettere tutto quanto insieme si può utilizzare il sistema di menù visto precedentemente...

20

Person & Address – Ricerca Esatta

- Ricerca di un contatto per cognome (*first name*)
- Problema facile e già visto
- Se i contatti sono:
  - ordinati → ricerca binaria
  - non ordinati → ricerca lineare
- Per semplicità si implementa la ricerca lineare...
- Si può utilizzare la strcmp come funzione di confronto fra stringhe...

21

Person & Address – Ricerca Esatta

In ingresso:

- Il cognome da cercare
- L'array in cui cercare
- Il numero di strutture effettivamente presenti nell'array

```
int findExactByFirstName(char firstName[50],
       PersonArray persons, int dim)
{
    int i;
    for (i = 0; i < dim; i++)
        if (strcmp(persons[i].firstName, firstName) == 0)
            return i;
    return -1;
}
```

In questo caso **ha veramente senso** effettuare una ricerca esatta?

22

Person & Address – Ricerca Parziale

- Ricerca di tutti i contatti il cui cognome inizia per...
- Non si fanno supposizioni sul tipo di ordinamento del contenitore dei contatti
  - però se fosse ordinato qualche ottimizzazione si potrebbe fare...
- Sostanzialmente, si vogliono "restituire" tutti i contatti il cui cognome... bla bla...

  → il problema è costruire una funzione che capisca se una stringa **`s`** comincia per una stringa **`start`**

23

Person & Address – Ricerca Parziale

Progetto della funzione startsWith

- Ingresso:
  - stringa **`s`** da verificare
  - stringa **`start`**
- Uscita:
  - true/false
- Per ogni carattere contenuto nella stringa start, se il carattere corrente non è il carattere nullo:
  - se il carattere corrente nella stringa start è diverso dal carattere corrispondente nella stringa s → terminare con insuccesso
  - altrimenti → continuare
  - se si arriva al termine naturale del ciclo → terminare con successo

24

Person & Address – Ricerca Parziale

```
boolean startsWith(char *s, char *start)
{
    int i;
    boolean ok = true;
    for (i = 0; start[i] != 0; i++)
        if (start[i] != s[i])
            return false;
    return true;
}
```

- Alternative utilizzando le funzioni sulle stringhe?
  - `strstr` cerca in una stringa la prima occorrenza di un'altra stringa data e restituisce un puntatore alla stringa trovata oppure `NULL` in caso di insuccesso
  - `char *strstr(char *string, char *strCharSet);`
  - es: `strstr("Nel mezzo del cammin...", "del")`
    → `"del cammin..."`

```
boolean startsWith(char *s, char *start)
{
    return strstr(s, start) == s;
    //return strstr(s, start) - s == 0;
}
```

25

Person & Address – Ricerca Parziale

- L'implementazione della funzione di ricerca diviene piuttosto semplice...

```
int findPartialByFirstName(char firstName[50], PersonArray
   inputPersons, int dim, PersonArray outputPersons)
{
    int personIndex, outputIndex = 0;

    for (personIndex = 0; personIndex < dim; personIndex++)
        if (startsWith(inputPersons[personIndex].firstName,
                    firstName))
            outputPersons[outputIndex++] =
                                        inputPersons[personIndex];
    return outputIndex;
}
```

26

Person & Address – Ricerca Parziale

- Però, però... Viene restituito un array di Person → un array di copie di Person
- È proprio necessario restituire delle copie?
- Gli originali rimangono, non vengono modificati dagli algoritmi visti → anziché restituire delle copie, è possibile restituire puntatori agli originali!

27

Person & Address – Ricerca Parziale

- Fra le definizioni:

```
typedef Person* PersonPtrArray[PERSONARRAYDIM];
```

- Il resto è molto simile

```
int findPartialByFirstName_Ptr(char firstName[50],
   PersonArray inputPersons, int dim,
   PersonPtrArray outputPersons)
{
    int personIndex, charIndex, outputIndex = 0;

    for (personIndex = 0; personIndex < dim; personIndex++)
        if (startsWith(inputPersons[personIndex].firstName,
                       firstName))
            outputPersons[outputIndex++] =
                                    &inputPersons[personIndex];
    return outputIndex;
}
```

28

Person & Address – Test

- Inserire alcuni contatti nell'array
- Effettuare una ricerca parziale per verificare il funzionamento degli algoritmi

```
...
num = findPartialByFirstName_Ptr("Zan", persons, 3,
  oPersonPtrs);

for (i = 0; i < num; i++)
{
    formatPerson(str, *oPersonPtrs[i]);
    printf("%s\n", str);
}
```

29

In generale...

- Passare puntatori a strutture è decisamente più efficiente che passare copie di strutture
- Ad es. le funzioni **`formatAddress`** e **`formatPerson`** potrebbero ricevere puntatori alle strutture corrispondenti
  - Come verrebbero modificate tali funzioni?
- Quando si ha anche fare con puntatori a strutture, ricordare di usare l'operatore **`->`**
  - **`p->field`** equivale a **`(*p).field`**

30

Menù V2

- Anziché usare due array separati (uno per le voci principali ed uno per le sotto-voci, sarebbe meglio usare un array di strutture in cui ogni struttura contiene una stringa che rappresenta la voce principale ed un array di stringhe che rappresenta le sotto-voci corrispondenti

```
typedef struct menuItemStruct
{
    char name[TEXTMAXDIM_2];
    char subMenus[MENUMAXDIM_2][TEXTMAXDIM_2];

} MenuItem;

MenuItem menuItems[MENUMAXDIM_2];
```

- L'interfaccia delle funzioni rimane la stessa, il codice ha una complessità circa equivalente → ...e sarebbe proprio un bell'esercizio...

31