C: Strutture

Fondamenti di Programmazione

Strutture

- Quando si usa una struttura ?
  - Quando abbiamo necessità di trattare un **insieme NON omogeneo** di dati !

Strutture

**Esempio*: memorizzare una data

```
int giorno, mese, anno;
```

Tre variabili per ogni data. Se serve un'altra data, dobbiamo dichiarare altre tre variabili

Le tre variabili sono logicamente collegate

Sarebbe utile poterle raggruppare! Usiamo le strutture

Strutture

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

Strutture: definizione

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

Questa rappresenta la **DEFINIZIONE** di una struttura.

La definizione permette di creare un **NUOVO TIPO**

Strutture: dichiarazione

```
struct data oggi;
struct data data_nascita;
```

Con la dichiarazione viene allocato lo spazio in memoria

Strutture: come accedere

Accedere ad una struttura:

`nome_variabile.nome_membro`

```
oggi.giorno=6;
oggi.mese=11;
oggi.anno=2018;
```

Strutture

Esempio:

```
#include <stdio.h>

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

int main() {
struct data oggi;

printf("Inserisci la data di oggi: ");
scanf("%d%d%d",&oggi.giorno,&oggi.mese,&oggi.anno);
printf("Oggi: %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Strutture: inizializzazione

```
#include <stdio.h>

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

int main() {
struct data oggi = {6, 11, 2018};

printf("\nOggi e' %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Strutture: inizializzazione

```c
#include <stdio.h>

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

int main() {
struct data oggi = {.giorno=6, .mese=11, .anno=2018};

printf("\nOggi e' %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Strutture: inizializzazione

```c
#include <stdio.h>

 struct data {
 int giorno;
 int mese;
 int anno;
} oggi = {.giorno=6, .mese=11, .anno=2018};

int main() {

printf("\nOggi e' %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Puntatori a strutture

```c
#include <stdio.h>
#define NOME 50
struct anagr
    {
        int matricola;
        char nome[NOME];
        char cognome[NOME];
    };

int main() {

    struct anagr studente;
    struct anagr *pointer;

    pointer=&studente;
```

Puntatori a strutture

```
    printf("\nNome studente: ");
    scanf("%s",studente.nome);
    printf("\nCognome studente: ");
    scanf("%s",studente.cognome);
    printf("\nMatricola: ");
    scanf("%d",&studente.matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",(*pointer).nome,
    (*pointer).cognome,(*pointer).matricola);
}
```

Puntatori a strutture

In alternativa: `pointer->nome` ...

```
    printf("\nNome studente: ");
    scanf("%s",studente.nome);
    printf("\nCognome studente: ");
    scanf("%s",studente.cognome);
    printf("\nMatricola: ");
    scanf("%d",&studente.matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",pointer->nome,
    pointer->cognome,pointer->matricola);
}
```

Puntatori a strutture

E ovviamente...

```
    pointer=&studente;

    printf("\nNome studente: ");
    scanf("%s",pointer->nome);
    printf("\nCognome studente: ");
    scanf("%s",pointer->cognome);
    printf("\nMatricola: ");
    scanf("%d",&pointer->matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",studente.nome,
    studente.cognome,studente.matricola);
}
```

Strutture contenenti puntatori

```
struct st
    {
        int *p1;
        int *p2;
    };
#include <stdio.h>
int main() {


    struct st st_pointers;
    int n1=10, n2;

    st_pointers.p1=&n1;
    st_pointers.p2=&n2;
    *st_pointers.p1=20;
    *st_pointers.p2=*st_pointers.p1 * 2;
    printf("%d %d",n1,n2);
}
```

Cosa stampa a video questo programma?

Liste concatenate lineari

Le liste lineari concatenate sono costituite da *serie di elementi omogenei* che occupano in memoria posizioni non adiacenti

Puntatori a strutture e *strutture contenenti puntatori* sono di fondamentale importanza per la creazione e gestione delle liste concatenate lineari

Liste concatenate lineari (statiche)

```c
#include <stdio.h>
    struct lista_valori {
        int num;
        struct lista_valori *next;
    };
int main() {
    struct lista_valori val1,val2;

    val1.num=10;
    val2.num=20;
    val1.next=&val2;
    printf("%d",val1.next->num);
}
```

Liste concatenate lineari

```c
#include <stdio.h>
    struct lista_valori
    {
        int num;
        struct lista_valori *next;
    };
int main() {
    struct lista_valori val1,val2,val3;

    val1.num=10;
    val2.num=20;
    val3.num=30;

    val1.next=&val2;
    val2.next=&val3;

    printf("%d ",val1.next->num);
    printf("%d ",val2.next->num);
}
```

Liste concatenate lineari

Alcune interessanti proprietà:

- eliminare elementi
- inserire elementi

Eliminare `val2` dalla lista:

```
val1.next = val2.next;
```

Inserire nuovo elemento (`val4`) tra `val2` e `val3` :

```
val4.next = val2.next;
```

```
val2.next = &val4;
```

Questo è possibile grazie al fatto che gli elementi di una lista non sono memorizzati in sequenza. *Cosa succede negli array?*

Liste concatenate lineari

- Puntatore esterno al primo elemento della lista
  - `struct lista_valori *punt_lista = &val1;`
- Puntatore a NULL (fine lista)
  - `val3.next = NULL;`

Esempio: scorri lista

```c
#include <stdio.h>
    struct lista_valori {
        int num;
        struct lista_valori *next;
    };
int main() {
    struct lista_valori val1,val2,val3;
    struct lista_valori *punt_lista;

    val1.num=10;
    val1.next=&val2;
    val2.num=20;
    val2.next=&val3;
    val3.num=30;
    val3.next=NULL;
    punt_lista=&val1;

    while(punt_lista != NULL) {
      printf("%d ",punt_lista->num);
      punt_lista=punt_lista->next;
    }
}
```

Liste concatenate lineari: NOTE

```
struct valori *punt_lista=&n1;
```

oppure

```
struct valori *punt_lista;
punt_lista=&n1;
```

Liste concatenate lineari: NOTE

- Liste statiche
- Liste dinamiche

Il programma ha necessità di allocare la memoria per ogni nuovo elemento della lista!

Allocazione dinamica della memoria