RIPASSO C
Parte 1

(Slides cortesemente concesse in uso dal Prof. Fraschini)

Le strutture

Tipi derivati

Strutture

- Permettono di usare un insieme di **variabili aggregate** con un unico nome
- 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

- Struttura per memorizzare una data

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

Strutture

- Struttura per memorizzare una data

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

CREA UN NUOVO TIPO

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

Definizione

Dichiarazione

Strutture

- Accedere ad una struttura:
  - **nome_variabile.nome_membro**

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

------->>Alloca lo spazio in memoria!

```
oggi.giorno=24;
oggi.mese=3;
oggi.anno=2017;
```

Strutture

```c
//Struttura data

#include <stdio.h>

int main(){

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

    printf("Inserisci la data di oggi come giorno mese anno: \n");
    scanf("%d%d%d",&oggi.giorno,&oggi.mese,&oggi.anno);
    printf("Oggi e': %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);

    return 0;
}
```

Strutture: inizializzazione

```c
//Struttura data

#include <stdio.h>

int main(){

    struct data
    {
        int giorno;
        int mese;
        int anno;
    };
    struct data oggi = {.giorno=24, .mese=3, .anno=2017};

    printf("Oggi e': %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);

    return 0;
}
```

Strutture: varianti

```
struct data
{
    int giorno;
    int mese;
    int anno;
} oggi,data_nascita;
```

Typedef

Lo scopo di **typedef** è di assegnare dei **nomi alternativi a dei tipi di dato esistenti**

- Evita dichiarazioni 'ingombrati'
- Rende il codice riutilizzabile più facilmente
- Rende il codice più leggibile

```
typedef tipo alias;
```

Typedef

Esempi

```
typedef int tipoPeso;
```

Typedef

Esempi

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

// struct data oggi
typedef struct data Tipodata;
Tipodata oggi;
```

Typedef

Esempi

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

// struct data oggi
typedef struct data Tipodata;
Tipodata oggi;
```

```c
typedef struct data2{
  int giorno;
  int mese;
  int anno;
} Tipodata2;
Tipodata2 oggi2;
```

Typedef

Esempi

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

// struct data oggi
typedef struct data Tipodata;
Tipodata oggi;
```

```
typedef int Tipodata3;
Tipodata3 oggi3;
oggi.giorno=21;
oggi3=100;
```

Vantaggi:

utilizzando alias posso assegnare una precisa **semantica** al tipo di dato

```
typedef int T_eta;
typedef int T_score;
```

Posso cambiare tipo di dato se cambiano le esigenze

```
typedef int T_eta;
typedef float T_score;
```

```c
#include <stdio.h>
// posso assegnare tipi compatibili - T_eta e  T_score sono sempre INT

int main() {
  typedef int T_eta;
  typedef int T_score;

  T_eta eta ;
  T_score score;

  score=10;
  eta=score;

  printf("eta, score: %d %f \n",eta, score);

  getchar();
  return 0;
}
```

```c
#include <stdio.h>
// struct data 1 e struct data2 sono tipi di dati DIVERSI
int main() {
  struct data1{
    int giorno;
    int mese;
    int anno;
  };
  typedef struct data1 Tipodata1;

  typedef struct data2{
    int giorno;
    int mese;
    int anno;
  } Tipodata2;

  Tipodata1 oggi1 ;
  Tipodata2 oggi2;

  oggi1.giorno=20;  oggi1.mese=5; oggi1.anno=2018;

  oggi2=oggi1;

```

```c
#include <stdio.h>
// problemi nell'uso di typedef : Maschero il tipo 'vero'

int main() {
  typedef struct data1{
    int giorno;
    int mese;
    int anno;
  } Tipodata1;

  typedef int Tipodata2;

  Tipodata1 oggi1 ;
  Tipodata2 oggi2;

  oggi1.giorno=20;
  oggi1.mese=5;
  oggi1.anno=2018;

  oggi2=100;
```

Array

Tipi derivati

Array

- Variabile strutturata: insieme di elementi dello stesso tipo.

- Tutti gli elementi hanno lo stesso nome (**nome_array**).
- Ogni elemento è identificato da un **indice**

nome_array[indice]

Esempio: voti[2] //permette di accedere al voto con indice 2

Il primo elemento di un array ha indice 0

voti[0] è il primo elemento dell'array

Array

x = voti[1];

assegna alla variabile x il contenuto dell'elemento con indice 1 dell'array voti

voti[1] = 2;

assegna il valore 2 all'elemento dell'array con indice 1

Array

- Permettono di sviluppare programmi concisi ed efficienti
- Esempio:

```
for(i=0; i<100; ++i)
        somma += voti[i];
```

ATTENZIONE:

- primo elemento ha indice 0
- ultimo elemento ha indice N – 1, dove N è la dimensione

Array: dichiarazione

- Dichiarazione: indicare il **tipo** e il **numero** massimo di elementi

int voti[20];

- dichiara un array che contiene 20 elementi di tipo int (senza dover dichiarare 20 variabili di tipo int!)
- possiamo usare indici che variano da 0 a 19
- ATTENZIONE: tentare di accedere ad un altro indice può **non generare un errore**!

Array: operazioni valide…

```
voti[2]=28;
voti[5]=voti[2];
voti[15]=voti[2] + voti[5];
voti[16]= voti[15] / 2;
++voti[15];
```

voti[100]=10; //!!!

Importante

int voti[10];
E' richiesta la conoscenza del fatto che la dimensione sarà 10
Se volessimo inserire 20 elementi?

Modifica noiosa e rischiosa!

Cambiare ogni occorrenza
Possibili diversi significati del numero 10

Utilizzo della direttiva **#define.** Si usa un nome simbolico.
#define NUMEROVOTI 10

int voti[NUMEROVOTI]

Array: inizializzazione

int voti[5] = {0,0,0,0,0};

int voti[5] = {0};

char lettere[4] = {'c', 'i', 'a', 'o'};

int voti[N];

for(int i=0; i<N; i++)

voti[i]=1;

```
1  #include <stdio.h>
2  #define DIM 3
3
4  int main()
5  {
6    int voti[DIM];
7    int i;
8
9    for(i = 0; i < DIM; ++i)
10     scanf("%d",&voti[i]);
11
12   for(i = 0; i < DIM; ++i)
13     printf("Voti[%d]=%d\n",i,voti[i]);
14
15   return 0;
16 }
17
```

Array: note

- Se la dimensione non è dichiarata gli inizializzatori la determinano

int vett[]={1,1,1,1};

L'array **vett** avrà 4 elementi.

- **In C non è possibile assegnare un array ad un altro! (poi vedremo perché)**

int v1[10],v2[10];

…

v1=v2; //ERRORE! ***occorre usare un ciclo for***

Array: note

- Il C non effettua alcun controllo sui limiti degli array
- In altre parole è possibile inizializzare un array di dimensione N con più di N valori senza avere alcun messaggio di errore in compilazione
- **È compito del programmatore garantire che tutti gli array siano abbastanza grandi da contenere ciò per cui sono stati creati**

Array e typedef

```
int V[10];
typedef int Vector_10[10];
// typedef int[10] Vector_10;  WRONG!
Vector_10 W;
V[0]=1;
W[0]=V[0];
```

Esempio: scrivere un programma che esegua la copia di un array in un nuovo array

```c
//copia tra arrays
#include <stdio.h>
#define DIM 100

int main()
{
    int arr1[DIM], arr2[DIM];
    int i, n;

    printf("Numero elementi del primo array:");
    scanf("%d",&n);

    printf("Inserisci i %d elementi:\n",n);
    for(i=0;i<n;i++)
    {
        printf("elemento - %d : ",i);
        scanf("%d",&arr1[i]);
    }
    /* Copia elementi*/
    for(i=0; i<n; i++)
    {
        arr2[i] = arr1[i];
    }
    /* Stampa elementi primo array   */
    printf("\nGli elementi del primo array sono:\n");
    for(i=0; i<n; i++)
        printf("%5d", arr1[i]);
    /* Stampa elementi secondo array */
    printf("\n\nGli elementi copiati nel secondo array sono:\n");
    for(i=0; i<n; i++)
        printf("%5d", arr2[i]);
    printf("\n\n");
}
```

Array e strutture

Approfondimenti ed esercizi

2. Array di strutture

```c
//struct_esami
#include <stdio.h>
#define DIM_1NOME 20
#define DIM_2NOME 20
#define DIM_STUD 100

int main()
{
    struct anagr
    {
        int matricola;
        char nome[DIM_1NOME];
        char cognome[DIM_2NOME];
    };

    struct anagr studente[DIM_STUD];
    int n_studenti;

    printf("Qunati studenti vuoi inserire?\n");
    scanf("%d",&n_studenti);

    printf("Inserisci studenti:\n");

    for(int i=0;i<n_studenti;++i)
    {
        printf("Studente - %d: \n", i+1);
        printf("Nome studente: \n");
        scanf("%s",studente[i].nome);
        printf("Cognome studente: \n");
        scanf("%s",studente[i].cognome);
        printf("Matricola: \n");
        scanf("%d",&studente[i].matricola);
    }

    printf("\n\nElenco studenti:\n");
    for(int i=0;i<n_studenti;++i)
    {
        printf("%s %s - matricola %d\n",studente[i].nome,studente[i].cognome,studente[i].matricola);
    }

    return 0;
}
```

2. Array di strutture

```c
struct anagr
{
    int matricola;
    char nome[DIM_1NOME];
    char cognome[DIM_2NOME];
};

struct anagr studente[DIM_STUD];
```

```c
for(int i=0;i<n_studenti;++i)
{
    printf("Studente - %d: \n", i+1);
    printf("Nome studente: \n");
    scanf("%s",studente[i].nome);
    printf("Cognome studente: \n");
    scanf("%s",studente[i].cognome);
    printf("Matricola: \n");
    scanf("%d",&studente[i].matricola);
}
```

2. Array di strutture

```
9     struct anagr
10    {
11        int matricola;
12        char nome[DIM_1NOME];
13        char cognome[DIM_2NOME];
14    };
15
16    struct anagr studente[DIM_STUD];
```

```
21    typedef struct anagr T_elem_anagrafe;
22    T_elem_anagrafe studente[DIM_STUD];
23
24    //struct anagr studente[DIM_STUD];
25
```