COESIONE ITALIA 21-27 PIEMONTE

Cofinanziato dall'Unione europea

REGIONE PIEMONTE

WEB DEVELOPER
Fondamenti di Programmazione

Massimo PAPA

LE STRUCT

LE STRUCT

- Abbiamo visto che l'array è un insieme di elementi omogenei tra di loro.
- Se dobbiamo gestire insieme di elementi non omogeni possiamo utilizzare gli array paralleli, ma non è molto agevole il loro utilizzo.
- Le STRUCT risolvono questo problema
- Infatti servono a contenere dati di tipo diverso come numeri, stringhe e anche array

LE STRUCT

- Generalmente il dato aggregato che la STRUCT descrive si chiama RECORD
- Il singolo elemento del RECORD si chiama CAMPO
- Esempio il record Persona può essere composto da
  - matricola (numero intero)
  - nome (stringa)
  - indirizzo (stringa)
  - dataNascita (stringa o record data)
  - codiceFiscale (stringa)
  - isConiugato (booleano)

LE STRUCT

- La dichiarazione di un RECORD avviene tramite la keyword `struct`

```
struct nomeStruttura {
   tipo1 campo1;
   tipo2 campo2;
   - - - - - - - -
   tipon campon;
};
```

LE STRUCT

- La dichiarazione di un RECORD avviene tramite la keyword `struct`

```
struct nomeStruttura {
   tipo1 campo1;
   tipo2 campo2;
   - - - - - - - -
   tipon campon;
};
```

LE STRUCT

- Esempio

```
struct Persona {
   int matricola;
   string nome;
   string indirizzo;
   string dataNascita;
   string codiceFiscale;
   bool isConiugato;
};
```

LE STRUCT

- Una volta definita la struttura e' come se avessimo definito un nuovo tipo di dato. Nell'esempio il tipo `struct Persona`.
- Utilizziamo la nuova struttura per definire le nostre variabili:

```
struct Persona dipendente;

struct Persona operaio;
```

- O anche:

```
Persona dipendente;

Persona operaio;
```

LE STRUCT

- I campi vengono referenziati con l'identificatore della variabile e con l'identificatore del campo stesso.
- Assegnazione:

```
dipendente.isConiugato = true;
```

- Lettura:

```
cout << dipendente.matricola;
```

STRUCT di STRUCT

- Riprendiamo l'esempio del campo dataNascita
- Possiamo definire la struct Data come segue:

```
struct Data {
   int gg;
   int mm;
   int aa
};
```

STRUCT di STRUCT

- Allora possiamo definire Persona come:

```
struct Persona {
   int matricola;
   string nome;
   string indirizzo;
   data dataNascita;
   string codiceFiscale;
   bool isConiugato;
};
```

STRUCT di STRUCT

- Con la definizione di data abbiamo definito un nuovo tipo che ci consente di gestire in maniera più agevole le date e tutte le funzionalità su intervalli di date.
- Assegnazione:

```
dipendente.dataNascita.aa = 2000;
```

- Lettura:

```
cout << dipendente.dataNascita.gg;
```

STRUCT e Funzioni

- Abbiamo capito che la definizione di una `struct` ci permette di definire un nuovo tipo di dato
- Avendo in mente questo concetto, possiamo andare a definire come parametri formali di una funzione le struct.
- Supponiamo di avere una funzione che calcoli il numero di giorni che intercorre tra due date.
- La funzione si potrebbe chiamare *intervallo*, accetta due date e restituisce un intero.

STRUCT e Funzioni

- Prototipo funzione intervallo:

```
int intervallo(Data dt1, Data dt2);
```

- oppure:

```
int intervallo(struct Data dt1, struct Data dt2);
```

- Nel caso precedente le strutture sono passate per valore.
- Passaggio per referenza:

```
int intervallo(Data& dt1, Data& dt2);
```

STRUCT e Funzioni

- Consideriamo una funzione che restituisce una struttura Data:

```
Data fun (Data dt1, int gg){
   Data valRitorno;
   . . .
   return valRitorno
};
```

STRUCT e Funzioni

- Chiamata della funzione fun:

```
int main(){
   Data data, dt1;
   . . .
   data = fun(dt1, 14);
   cout << data.aa;
   . . .
   return 0
};
```

STRUCT e Funzioni

Per ulteriori esempi consulta le successive slide

21_Esempio passaggio di parametri