Corso sul linguaggio C++

Modulo 3

2 - Struttura dati **struct**

Prerequisiti

- Programmazione elementare in C++
- Uso di funzioni

Introduzione

In questa lezione fissiamo l'attenzione sulla necessità di un altro tipo di variabile strutturata che prende il nome di **struct**.

Impariamo come si dichiara una **struct** in C++ e quale possa essere il suo utilizzo in diversi ambiti.

Impariamo anche un primo esempio di composizione di strutture dati per generare strutture astratte più complesse.

Argomenti

- La **struct**
- Dichiarazione di una **struct**
- Inizializzazione di una **struct**
- Il costrutto **typedef**
- Utilizzo della **struct**
- La **struct** come parametro
- La **struct** come valore di ritorno
- Array di **struct**

Informazioni generali

Così come abbiamo fatto per array e matrici, vediamo anche per la struct come si dichiara in C++, come si possano accedere ai dati in essa contenuti e quali sono le situazioni reali che essa consente di modellizzare facilmente.

La **struct**

Quando dobbiamo trattare con un insieme di variabili che appartengono a tipi di dato generalmente diversi, ma che sono tra loro logicamente correlate, si ricorre al modello di dato strutturato detto **struct**.

Anche la **struct**, come l'array, è un dato strutturato in quanto contiene nel suo interno variabili di tipo più semplice, che in questo caso prendono il nome di **campi** o **attributi** o **membri.**

Dichiarazione di una **struct**

In base a quanto detto, la sintassi dichiarativa di una variabile di tipo **struct** è la seguente:

struct *ident* // nome della struct
{ *dichiarazioni membri*
} *iden*; // nome della variabile di tipo struct

struct è la parola chiave per la la struttura e specifica il tipo di dato

Esempio:

```
struct personale
{  string cognome;
   int codice;
   bool presenza;
} dipendente;
```

L'occupazione di memoria di una **struct** è data dalla *somma dell'occupazione in byte* dei suoi membri.

Dichiarazione di una **struct**

Descriviamo la composizione della struttura attraverso un altro esempio:

```
struct francobollo
{   int anno;
    string tema;
    float prezzo;
} collezione;
```

francobollo è **l'identificatore** scelto dal programmatore per il tipo di dato. Rappresenta il nome della **struct**, ma non può essere utilizzato nel programma perché NON è la variabile. Prende anche il nome di **etichetta**.

collezione è la variabile di tipo **struct** che sarà composta dai vari membri indicati

In questo tipo di dichiarazione l'identificatore "francobollo" può mancare (*dichiarazione anonima*)

Dichiarazione di una **struct**

Un modo alternativo per la definizione delle variabili di tipo **struct** è il seguente:

```
struct francobollo
{   int anno;
    string tema;
    float prezzo;
};
struct francobollo collezione;
```

La definizione delle variabili è esterna alla dichiarazione della **struct**.

Inizializzazione di una **struct**

Per selezionare all'interno della **struct** un membro, si utilizza l'operatore "." (punto) che prende il nome di **selettore.**

```
struct francobollo
{   int anno;
    string tema;
    float prezzo;
};
struct francobollo collezione;
…….
collezione.anno=35;
…….
```

Pertanto, ad esempio, per assegnare il valore 35 al membro *anno* si scrive come indicato

Inizializzazione di una **struct**

Una variabile **struct** può essere inizializzata da programma nel modo indicato

```
#include <iostream>
#include <cstdlib>
struct francobollo
{   int anno;
    string tema;
    double prezzo;
 };
int main()
{  struct francobollo esemplare={1987,"A.Manzoni",27.5};
    cout<<"anno: "<<esemplare.anno<<"\n";.
    cout<<"Tema: "<<esemplare.tema<"\n";
    cout<<"Prezzo: "<<esemplare.prezzo<<"\n";
    system("PAUSE");
    return 0;
}
```


Il costrutto **typedef**

Il costrutto **typedef** viene associato spesso ad una **struct**. in modo da abbreviare le dichiarazioni successive.

Esempi:

```
typedef struct data
{
    int giorno;
    string mese;
    int anno;
};
data ieri, oggi, domani;
```

```
typedef struct
{
    int giorno;
    string mese;
    int anno;
} data;
data  ieri, oggi, domani;
```

Utilizzo della **struct**

Il tipo **struct** può essere:

- passato ad una funzione come parametro;
- restituito come tipo di una funzione.

La **struct** come parametro

ATTIVITA': data la dichiarazione seguente, scrivere il prototipo di una funzione *inserisci_anno()* che riceva una variabile *c* di tipo *francobollo* e l'anno *a* da inserire in essa.

```
struct francobollo
{   int anno;
    string tema;
    double prezzo;
};
```

Dovendo modificare *c* è necessario che questa venga passata per indirizzo. Il tipo della funzione è ovviamente **void**.

```
void inserisci_anno (struct francobollo &c,  int a);
```

ATTIVITA': scrivere la definizione della funzione *inserisci_anno()*.

```
void inserisci_anno (struct francobollo &c, int a)
{  c.anno=a;  }
```

La **struct** come parametro

```
#include <iostream>
#include <cstdlib>
struct francobollo
{   int anno;
    string tema;
    double prezzo;  };
void inserisci_anno (struct francobollo &c,  int a);
int main()
{   int v_anno;
    struct francobollo esemplare;
    cout<<"\nAnno di emissione: ";     cin>>v_anno;
    inserisci_anno (esemplare, v_anno);
    cout<<"Inserito per l'anno "<<esemplare.anno<<"\n";
    system("PAUSE");
    return 0;  }
void inserisci_anno (struct francobollo &c, int a)
{  c.anno=a;  }
```

ATTIVITA': scrivere il codice completo dell'esercizio.

La **struct** come valore di ritorno

ATTIVITA': data la dichiarazione seguente, scrivere il prototipo di una funzione *inserisci()* che legga i 3 campi ed abbia valore di ritorno di tipo *francobollo*.

```
struct francobollo
{   int anno;
    string tema;
    double prezzo;
};
```

La funzione ha argomento **void** perché il dato di tipo **struct** letto viene restituito come valore di ritorno.

```
struct  francobollo inserisci ();
```

ATTIVITA': scrivere la definizione della funzione *inserisci()*.

La **struct** come valore di ritorno

```
struct francobollo inserisci ()
{  struct francobollo f;
    cout<<"\nAnno..: ";
    cin>>f.anno;
    cout<<"Tema..: ";
    cin>>f.tema;
    cout<<"Prezzo: ";
    cin>>f.prezzo;
    return f;
 }
```

In questo esempio si crea una variabile locale *f* di tipo **struct** *francobollo* per leggere i dati.

Al termine della lettura *f* viene restituita al programma chiamante

La **struct** come valore di ritorno

```
#include <iostream>
#include <cstdlib>
struct francobollo
{   int anno;
    string tema;
    double prezzo;
};
struct francobollo inserisci ();

int main()
{  struct francobollo esemplare;
   esemplare=inserisci ();
   cout<<"Inserito“<<endl;
   system("PAUSE");
   return 0;
}
…segue
```

ATTIVITA': scrivere il codice completo dell'esercizio.

```
…segue
struct francobollo inserisci ()
{  struct francobollo f;
   cout<<"\nAnno..: ";    cin>>f.anno;
   cout<<"Tema..: ";    cin>>f.tema;
   cout<<"Prezzo: ";    cin>>f.prezzo;
   return f;
 }
```

Array di **struct**

```
#include <iostream>
#include <cstdlib>
struct francobollo
{   int anno;
    string tema;
    double prezzo;
};
int main()
{  struct francobollo esemplare[100];
   ……
   ……
   ……
   ……
   system("PAUSE");
   return 0;
}
```

Un tipo **struct** può essere componente di un array. Sempre con la medesima dichiarazione di *francobollo*, l'array può essere dichiarato come indicato. Esso rappresenta un **array di strutture** detto anche **tabella**

Array di **struct**

```
struct francobollo
{   int anno;
   char tema[20];
   double prezzo;
};
int main()
{  struct francobollo esemplare[100]=  {
                                       {1987,"A.Manzoni",27.5 },
                                       {1987,"B. Franklin",32.7}
                                       };
   ……..
   ……..
   system("PAUSE");
   return 0;  }
}
```

L'inizializzazione avviene con le parentesi graffe come per gli array; i vari elementi di tipo **struct** sono anche essi racchiusi in una ulteriore coppia di parentesi graffe e separati da virgola.

Array di **struct**

ATTIVITA': scrivere un programma che servendosi di un vettore *collezione[ ]* di *MAXELEMENTI* di tipo *francobollo* effettui:
- il caricamento del vettore;
- la stampa del vettore;
- la stampa dei francobolli di un dato *anno*, ripetuta fino a quando *anno* vale 0.

Scrivere l'algoritmo richiesto.

```
Inizio
    Per i=0 a MAXELEMENTI-1 fai inserisci();
    Per i=0 a MAXELEMENTI-1 fai stampa_dati(i);
    Ripeti   Leggi (anno);
             Per i=0 a MAXELEMENTI-1 fai
                Se  (collezione[i].anno==anno)
                    stampa_dato(collezione[i]);
    Fintantochè (anno!=0);
Fine.
```

Dobbiamo implementare le funzioni indicate.

Array di **struct**

ATTIVITA': data la dichiarazione seguente, scrivere il frammento di codice per caricare l'array con *MAXFRANCOBOLLI* elementi, facendo uso della funzione *inserisci()* dell'esercizio precedente.

```
#define MAXFRANCOBOLLI 3
struct francobollo
{   int anno;
    string tema;
    double prezzo;
};
struct francobollo collezione[MAXFRANCOBOLLI];
…

for (int i=0; i<MAXFRANCOBOLLI; i++)
      collezione[i]=inserisci();
```

Per semplicità dichiariamo il vettore *collezione[ ]* come globale

Ogni elemento del vettore *collezione*[ ] viene caricato con una **struct** letta da input.

Array di **struct**

ATTIVITA':scrivere la funzione *stampa_dati(...)* che ricevendo come parametro l'indice *n* del vettore, stampa l'elemento in posizione *i*.

```
void stampa_dati (int n)
{  cout<<collezione[n].anno<<endl;
   cout<<collezione[n].tema<<endl;
   cout<<collezione[n].prezzo<<endl;
   cout<<endl;
}
```

Essendo *collezione[ ]* globale la struttura della funzione è questa

Array di **struct**

ATTIVITA':scrivere la parte di codice che consente la stampa filtrata dei soli elementi *francobollo* aventi il campo *anno* richiesto.

```
do
  {  cout<<"Visualizza per anno";
     cout<<"Immetti anno: ";
     cin>>anno;
     for (int i=0;i<MAXFRANCOBOLLI; i++)
       if (collezione[i].anno==anno)
          stampa_dato(collezione[i]);
  } while (anno);
```

La condizione di uscita si verifica quando si immette il valore 0 per la variabile *anno*.

Cosa ho imparato

- Cosa è il tipo **struct** e in quali casi può essere utilizzato
- Come si dichiara una variabile di tipo **struct**
- Come si accede ai membri di una **struct**
- Il tipo **struct** può essere usato mediante **typedef**
- Una variabile di tipo **struct** può essere utilizzata sia come parametro di una funzione (passabile per valore o per indirizzo), che come valore di ritorno.
- Come si può modellizzare una tabella mediante array e **struct**

Cosa ho imparato a fare

- Scegliere il tipo struct in base al problema
- Dichiarare variabili di tipo **struct**, utilizzare il tipo **struct** mediante una **typedef**, creare un array di **struct** (tabella)
- Scrivere semplici applicazioni facenti uso di **struct** e di tabelle
- Scrivere funzioni che hanno **struct** come parametri o come valori di ritorno

Terminologia

- Campi
- Struttura dati
- **struct**
- Selettore

Altre fonti di informazione

- J. Purdum, C – ed. Jackson
- Romagnoli-Ventura, C/C++ - Ed. Petrini
- A.Lorenzi-D.Rossi, Il linguaggio C++ - ed. ATLAS