Laboratorio di programmazione

Lezione VI

Tatiana Zolo

zolo@cs.unipr.it

LE STRUCT

- Dichiarazione (forma generale):

```
struct nome_struttura {
  tipo var1;
  tipo var2;
  ...
  tipo vark;
} variabili_struttura;
```

(soltanto una variabile struttura $\Longrightarrow$ `nome_struttura` non necessario).

Ora posso creare delle variabili di tipo `nome_struttura`:

```
nome_struttura var_struttura;
```

LE STRUCT

- Dichiarazione (forma generale):

```
struct nome_struttura {
  tipo var1;
  tipo var2;
  ...
  tipo vark;
} variabili_struttura;
```

(soltanto una variabile struttura $\Longrightarrow$ `nome_struttura` non necessario).

Ora posso creare delle variabili di tipo `nome_struttura`:

```
nome_struttura var_struttura;
```

- Accesso ai membri: tramite l'operatore "punto".

```
nome_struttura.var1 = ...;
tipo t = nome_struttura.var2;
```

LE STRUCT

- Assegnamento di strutture: solo se sono dello stesso tipo.

LE STRUCT

- Assegnamento di strutture: solo se sono dello stesso tipo.
- Array di struct: si definisce una struttura e poi si dichiara un array di quel tipo. Esempio:

```
struct studenti {
  char nome[50];
  char cognome[50];
  int anno_immatricolazione;
};
studenti elenco_studenti[100];
```

Per accedere ad una struttura specifica $\Longrightarrow$ indicizzare il nome della struttura: `cout << elenco_studenti[2].cognome`
stampa il cognome memorizzato nella variabile membro `cognome` dello studente i cui dati sono nella 3° struttura dell'array (l'indice dell'array parte sempre da 0!).

LE FUNZIONI

Ogni funzione è composta da 4 parti (che insieme costituiscono la definizione della funzione):

1. tipo di ritorno;
2. nome della funzione;
3. lista dei parametri;
4. corpo della funzione.

Le prime tre parti insieme rappresentano il prototipo della funzione o la dichiarazione della funzione.

Invocazione funzione `f()` $\Longrightarrow$ controllo del programma ad `f()` e sospensione dell'esecuzione della funzione attiva.

Fine esecuzione di `f()` (ultima istruzione oppure *istruzione di ritorno*) $\Longrightarrow$ la funzione sospesa riprende l'esecuzione.

PROTOTIPO DI UNA FUNZIONE

1. Tipo di ritorno: può essere
   - ➜ tipo predefinito (es.`int`);
   - ➜ tipo composto (es.`double*`, cioè puntatori);
   - ➜ tipo definito dall'utente (es. struct, classi);
   - ➜ `void`, cioè la funzione non restituisce alcun valore.

   Non si possono usare come tipi di ritorno un tipo funzione oppure un tipo array predefinito.

   C++ standard: il tipo di ritorno non può essere omesso $\Longrightarrow$ errore a tempo di compilazione.

PROTOTIPO DI UNA FUNZIONE

1. Tipo di ritorno: può essere
   - ➜ tipo predefinito (es.`int`);
   - ➜ tipo composto (es.`double*`, cioè puntatori);
   - ➜ tipo definito dall'utente (es. struct, classi);
   - ➜ `void`, cioè la funzione non restituisce alcun valore.

   Non si possono usare come tipi di ritorno un tipo funzione oppure un tipo array predefinito.

   C++ standard: il tipo di ritorno non può essere omesso $\Longrightarrow$ errore a tempo di compilazione.
3. Lista dei parametri:
   - ➜ in una definizione di funzione un nome di parametro consente di accedere al parametro nel corpo della funzione;
   - ➜ in una dichiarazione di funzione il nome del parametro non è necessario (se c'è può essere diverso da quello della def. di funz.).

   Controllo di tipo dei parametri al momento della compilazione.

Passaggio degli argomenti

- Passaggio per valore:

```
int nome_funz(int n) {
  n = n + 1;
  return n;
}
```

  Chiamata: `n = 0; m = nome_funz(n);`

  Risultato: $n = 0, m = 1$

- Passaggio per riferimento:

```
int nome_funz(int& n) {
  n = n + 1;
  return n;
}
```

  Chiamata: `n = 0; m = nome_funz(n);`

  Risultato: $n = 1, m = 1$

PARAMETRI ARRAY

In C++ un array non è mai passato per valore, ma come un puntatore al suo primo elemento. Conseguenze:

- ➜ le modifiche sono effettuate sull'array stesso e non su una copia locale;
- ➜ la dimensione di un array non fa parte del suo tipo parametro.

es.`array_candidati.cpp`.

ESERCIZI

1. Struct: creare una struct "persona" (il nome scieglietelo pure voi, ma sensato) con due membri: nome e numero di telefono. Scrivere un programma che, una volta riempite con i dati necessari tre struct "persona", dia le seguenti 3 possibilita':
   1. conoscere il nome a partire da un numero di telefono;
   2. conoscere il numero di telefono a partire da un nome;
   3. uscire dal programma.

   (`nome_telefono.cpp`).
2. Array di struct e funzioni: leggere 'n' struct di persone ('n' inserito dall'utente) con due campi, nome ed eta', e restituire il nome della piu' giovane o di una delle piu' giovani. Utilizzare una funzione per la lettura dei dati delle struct e una che restituisca l'indice della persona selezionata.
   (`piu_giovane_array.cpp`).

ESERCIZI

3. Array di struct e funzioni: crea un array di struct "studente" (cognome, array contenente i voti degli esami). Presenta all'utente un menu' con varie possibilita':
   - ➜ media (in trentesimi);
   - ➜ media (in 110);
   - ➜ voto piu' alto;
   - ➜ voto piu' basso;
   - ➜ fine programma.

   Realizza ciascuna operazione con una funzione.

   Possibili estensioni dell'esercizio: altre operazioni possibili sono vedere quante volte e' stato preso un certo voto; fare un grafico dell'andamento dei voti (es. con una matrice di caratteri). Ampliare la struct "studente" inserendo altri dati: nome materia, data esame, eventuale lode, ecc. (`voti_esami.cpp`).