Università degli Studi
di Cassino

Corso di Fondamenti di Informatica

Tipi strutturati: Strutture typedef

Anno Accademico 2009/2010

Francesco Tortorella

Le strutture

- Gli array permettono la sola aggregazione di variabili dello stesso tipo.
- Quindi non rispondono all'esigenza di aggregare dati di tipo diverso, come accade, p.es., per i dati anagrafici (nome, cognome, anno nascita).
- Per questo scopo, il C++ mette a disposizione le **strutture**.

Le strutture

- Una struttura è una collezione di elementi denominati **campi**, ognuno dei quali può contenere un dato di tipo diverso.
- Ogni campo può essere di tipo atomico (`int`, `float`, ...) o strutturato (array, stringa).
- Quando si dichiara una struttura vanno quindi precisati quali siano i campi componenti e a quale tipo appartengano.

Il costruttore di tipi `struct`

- La dichiarazione di una struttura avviene tramite il costruttore **`struct:`**

```
struct nome_struttura {
        tipo_campoA nome_campoA;
        tipo_campoB nome_campoB;
        …
        tipo_campoX nome_campoX;
};
```

- I campi sono identificati con etichette differenti, non con indici (come accade per gli array), visto che appartengono a tipi differenti.

Esempio di dichiarazione

```
struct Studente {
        char nome[32];
        char cognome[32];
        int matricola;
        char cds[8];
};
```

- La dichiarazione della struttura crea un nuovo tipo con identificatore **`studente`**, che può essere utilizzato nel resto del programma.

Definizione di variabili struttura

- Le variabili struttura possono essere definite usando l'identificatore introdotto nella dichiarazione:
  `studente s1,s2,s3;`
- La definizione è anche possibile nell'ambito della dichiarazione della struttura:

```
struct Studente {
      char nome[32];
      char cognome[32];
      int matricola;
      char cds[8];
} s1, s2, s3;
```

Si dichiara la struct studente

Si definiscono le variabili s1, s2, s3

Inizializzazione di variabili struttura

- Come gli array, una variabile struttura può essere inizializzata all'atto della definizione, facendo attenzione a che i valori siano forniti nell'ordine corretto:

```
Studente s1 = {"Paolino",
      "Paperino",
      1234,
      "LINFTLC"};
```

Accesso ai campi delle variabili struttura

- Per accedere in lettura o in scrittura ad un singolo campo della struttura si utilizza l'operatore punto (.): *nome_var_struttura.nome_campo*
- Ogni campo della struttura è una variabile (del proprio tipo) a tutti gli effetti:

```
Studente s1 = {"Paolino",
       "Paperino",
       1234,
       "LINFTLC"};
int a;
char s[32];

a=s1.matricola;
strcpy(s,s1.cds);
s1.matricola=4321;
```

Inizializzazione di variabili struttura

- Al di fuori delle definizioni, l'inizializzazione deve avvenire su ogni campo:

```
Studente s1;
strcpy(s1.nome,“Paolino”);
strcpy(s1.cognome,“Paperino”);
s1.matricola = 1234;
strcpy(s1.cds, “LINFTLC”);
```

- In questo caso, l'assegnazione dei campi può avvenire in un ordine qualunque.

Assegnazione tra variabili struttura

- Diversamente dagli array, le variabili struttura possono essere utilizzate direttamente in un'assegnazione:

```
Studente s1 = {“Paolino”,
      ”Paperino”,
      1234,
      “LINFTLC”}, s2;

s2 = s1;
```

Array di struct

- Gli array di strutture sono utili per gestire particolari elenchi:

```
struct Studente {
      char nome[32];
      char cognome[32];
      int matricola;
      char cds[8];
};


Studente iscritti[100];
```

Le strutture come parametri

- In C++ è possibile passare strutture come parametri ai sottoprogrammi sia per valore che per riferimento (diversamente dagli array).
- Inoltre è possibile che la funzione restituisca direttamente una struttura tramite il suo nome.

Le strutture come parametri

- Esempio: lettura di una variabile struttura

```
// SOLUZIONE 1
void leggi_stud1(Studente& s){

        cin >> s.nome;
        cin >> s.cognome;
        cin >> s.matricola;
        cin >> s.cds

        return;
}
```

```
// SOLUZIONE 2
Studente leggi_stud2(){
        Studente s;

        cin >> s.nome;
        cin >> s.cognome;
        cin >> s.matricola;
        cin >> s.cds

        return s;
}
```

Quali sono le differenze ?

Problema

- Scrivere un programma che gestisca un elenco di studenti appartenenti a diversi corsi di studi. In particolare, realizzi le operazioni di:
- lettura iniziale dell'elenco;
- creazione e stampa del sottoelenco formato dagli studenti appartenenti ad uno stesso corso di studi.

typedef

- La dichiarazione di una struttura introduce un nuovo tipo a quelli presenti nel linguaggio.
- Il C++ permette l'introduzione di nuovi tipi basati su tipi già esistenti tramite la parola chiave **`typedef`** che ha sintassi:

  `typedef tipo_esistente nome_nuovo_tipo`;

  dove **`tipo_esistente`** è un tipo C++ (fondamentale o strutturato) mentre **`nome_nuovo_tipo`** è il nuovo nome per quel tipo.
- Esempio:
  `typedef double Doppio`;
  `typedef float Punto[2]`;

typedef

- I nuovi nomi possono essere utilizzati ovunque possono essere utilizzati i nomi precedenti:
  ```
  Doppio x,y;
  Punto p1, p2, p3;
  ```
- Sotto tutti i punti di vista i due nomi si riferiscono allo stesso tipo.
- Questo strumento può essere utile per definire un alias di un tipo frequentemente usato in un programma oppure per definire tipi che potrebbero essere modificati in versioni successive del programma (es. **`float`** $\rightarrow$ **`double`**).

Confronto tra array e struct nella progettazione dei tipi

- Quando si va a definire un nuovo tipo strutturato che potrebbe essere organizzato tramite un array, potrebbe piuttosto essere conveniente utilizzare una struttura.
- Come esempio supponiamo di voler definire un tipo per gestire numeri complessi.

Uso dell'array

```
typedef  float Complesso[2];

// definizione
Complesso x,y,z;

// accesso alla parte reale
x[0] = 1.3;

// accesso al coefficiente dell'immaginario
x[1] = 2.1;

// espressioni
modulo=sqrt(x[0]*x[0]+x[1]*x[1]);
```

Achtung: il tipo `Complesso` è comunque un array

Uso della struttura

```
struct Complesso{
       float Re;
       float Im;
};

// definizione
Complesso x,y,z;

// accesso alla parte reale
x.Re = 1.3;

// accesso al coefficiente dell'immaginario
x.Im = 2.1;

// espressioni
modulo=sqrt(x.Re*x.Re+x.Im*x.Im);
```