Inf Gen II

INFORMATICA GENERALE II

Ingegneria delle Telecomunicazioni
Università di Trento

Marco Roveri

roveri@irst.itc.it

Strutture e tipi elementari

AA 2006/2007

MR

Inf Gen II

Il problema

- Supponiamo di dover gestire una rubrica telefonica. Ogni elemento della rubrica è caratterizzato da un *nome,* l'anno di nascita della persona, un *numero di telefono*.
- Se abbiamo un solo elemento nella rubrica, il problema è semplice.
  - Usiamo una variabile per ogni informazione che vogliamo memorizzare.

```
#include <iostream>
using namespace std;
void stampapersone(char *, int, char *);
int main () {
     char * name = “Mario Rossi”;
     int   year = 1965;
     char * phone = “0461314326”;
     stampapersone(name, year, phone);
}
void stampapersone(char * name, int year, char * phone) {
      cout << “Nome = “ << name << endl;
      cout << “Anno = “ << year << endl;
      cout << “Phone = “ << phone << endl;
}
```

AA 2006/2007

MR

2

Inf Gen II

Il problema

- Se la rubrica contiene N elementi?
  - Potremmo usare un array per ogni informazione da memorizzare. Tutti gli elementi con indice *i* si riferiscono alla stessa entry nella rubrica.

```
#include <iostream>
using namespace std;
void stampapersone(char *, int, char *);
int main () {
     char * names[] = {"Mario Rossi", "Paolo Bianchi", "Andrea Neri"};
     int   years[] = {1965, 1970, 1980};
     char * phones[] = {"0461314326", "046131458", "0461314328"};

     for(int i = 0; i < 3; i++) {
          cout << "------------- " << i " << "---------------" << endl;
          stampapersone(names[i], years[i], phones[i]);
     }
}
void stampapersone(char * name, int year, char * phone) {
      cout << "Nome = " << name << endl;
      cout << "Anno = " << year << endl;
      cout << "Phone = " << phone << endl;
}
```

AA 2006/2007

MR

Inf Gen II

Il problema

- Il programma precedente è corretto, ma…
  - Leggibilità:
    - Non risulta subito chiaro che le tre variabili sono attributi dello stesso "oggetto".
  - Modificabilità:
    - Se vogliamo aggiungere la data di nascita, bisogna aggiungere una variabile, e poi modificare tutte le chiamate di funzione.
- Il primo problema può essere causa di errori.
  - Esempio: invoco la funzione stampapersona commettendo l'errore di passare il nome di una persona, il cognome di un'altra e l'età di una altra persona ancora.

```
stampapersona(names[1], years[2], phones[3]);
```

- Il secondo problema può rendere difficile modificare il codice, e questo può a sua volta portare a errori se il codice non viene modificato nel modo giusto.

AA 2006/2007

MR

La soluzione

- Definire una variabile di tipo persona, che contiene al suo interno tutti i dati di una persona, ossia nome, anno di nascita, e numero di telefono.
- **Non** possiamo usare un **unico** array, perchè ci troviamo a manipolare dati eterogenei (stringhe, interi, …).
- Il C++ mette a disposizione diversi costrutti per definire **nuovi tipi di dato** secondo le varie specifiche del caso.
- Il costrutto più complesso e completo è il costrutto **class.** Questo costrutto consente di definire non solo i **valori** che il nuovo tipo può assumere, ma anche le **operazioni** che si possono effettuare su di esso.
- In questo corso prenderemo in considerazione solo il modo più semplice e conseguentemente meno espressivo di definire un nuovo tipo di dato complesso, utilizzando il costrutto **struct**.

Costrutto struct

- Una struttura è una *n*-upla ordinata di elementi, detti *membri* o *campi*, ciascuno dei quali ha uno specifico *tipo*, un *nome* e un *valore.*
- Sintassi (definizione di un nuovo tipo):

```
struct new_struct_id {
    tipo1 campo1;
    …
    tipon campon; };
```

Ricordarsi di inserire il “;” finale.

- I campi di una struct possono essere rappresentati da un tipo fondamentale o da un qualsiasi altro tipo precedentemente definito.

Inf Gen II

Esempio

```
// Dichiarazione di una struttura per rappresentare la data
struct Data {
  int giorno;
  int mese;
  int anno;
};

// Dichiarazione di una struttura per rappresentare i dati
// di una persona
struct Persona {
   char * name;
   Data   year;
   char * phone;
};
```

Tipi fondamentali

Tipo precedentemente dichiarato

AA 2006/2007

MR

Inf Gen II

Convenzioni

- In C++, ma anche in altri linguaggi come Java, gli oggetti (strutture e classi) hanno nomi capitalized (ovvero la prima lettera dell'identificativo maiuscola).
- Questo *non* è richiesto dal C++, ma è buona norma seguire questa convenzione per aumentare la leggibilità del codice.

AA 2006/2007

MR

Inf Gen II

Definizione di una variabile di tipo struct

- Può essere definita nei seguenti modi:
  - *New_struct_id var_id*; **-- Modo 1**
    - Qui assumiamo che *New_struct_id* sia stato definito precedentemente.
  - **struct** *New_struct_id* **{ -- Modo 2**
    $tipo_1$ $campo_1$;
    ...
    $tipo_n$ $campo_n$; } *var_id*;
  - **struct { -- Modo 3**
    $tipo_1$ $campo_1$;
    ...
    $tipo_n$ $campo_n$; } *var_id*;
    - quest'ultima è detta struttura anonima

AA 2006/2007 MR 9

Inf Gen II

Esempio

```
// Modo 1
struct Data {
  int giorno;
  int mese;
  int anno;
};

int main() {
  Data d;
  …
}
```

```
// Modo 2


int main() {
  struct Data {
    int giorno;
    int mese;
    int anno;
  } d;
  …
}
```

```
// Modo 3

int main() {
  struct {
    int giorno;
    int mese;
    int anno;
  } d;
  …
}
```

AA 2006/2007 MR 10

Definizione di una variabile di tipo struct

```
// Modo 1
struct Data {
   int giorno;
   int mese;
   int anno;
 };

 int main() {
   Data d;
   …
  }
```

- Il tipo nuovo Data dichiarato in questo modo è visibile
  - in tutto il file in cui la dichiarazione di Data occorre;
    - ma dalla posizione in cui la dichiarazione occorre in poi.
  - se la dichiarazione è in un file nome.h (header file) allora è visibile in tutto il file che include name.h;
    - ma anche in questo caso dalla posizione in cui l'inclusione del file occorre.
- In qualunque blocco nel file seguente alla dichiarazione o all'inclusione si possono dichiarare variabili di tipo Data.

Inf Gen II

AA 2006/2007

MR

11

Definizione di una variabile di tipo struct

```
// Modo 2

 int main() {
   struct Data {
     int giorno;
     int mese;
     int anno;
    } d;
    …
  }
```

- Il tipo nuovo Data definito in questo modo 2 è visibile solo nel blocco in cui il tipo è dichiarato.
  - Si possono dichiarare solo all'interno del blocco in cui Data è dichiarato diverse variabili di tipo Data.

Inf Gen II

AA 2006/2007

MR

12

Inf Gen II

Definizione di una variabile di tipo struct

```
// Modo 3

int main() {
  struct {
    int giorno;
    int mese;
    int anno;
   } d;
   …
 }
```

- Il tipo anonimo non consente di dichiarare altre variabili dello stesso tipo.

```
struct { int j;} k;
struct {int j;} m;
```

  – In questo caso le variabili k e m possono essere considerate dal compilatore variabili di tipo diverso.

AA 2006/2007

MR

Inf Gen II

Accesso agli elementi di una struttura

- Se **s** è una struttura e **field** è un identificatore di un campo
  – allora **s.field** denota il campo della struttura
- Esempio:

```
#include <iostream>
using namespace std;
struct complex { double re, im; };

int main() {
 complex c;
 c.re = 2.5;
 c.im = 3;
 cout << “Parte reale = “ << c.re << endl;
 cout << “Parte immaginaria = “ << c.im << endl;
}
```

AA 2006/2007

MR

Inf Gen II

Strutture e puntatori

- Di una variabile di tipo struttura si può determinare sia l'indirizzo della struttura che l'indirizzo di ogni suo elemento.
- Similmente si può sapere lo spazio di memoria occupata dalla struttura e dai suoi singoli elementi.
- Una struttura definisce un tipo, è quindi possibile definire dei puntatori a tali strutture.

AA 2006/2007

MR

15

Inf Gen II

Strutture e memoria

```
#include <iostream>
using namespace std;
struct Data {
   int giorno;
   int mese;
   int anno;
};

int main() {
   Data d;
   Data * pd;

   cout << “Size of Data = “ << sizeof(Data) << “ = “ Size of d = “ << sizeof(d) << endl;
   cout << “Indirizzo di d = “ << &d << endl;
   cout << “Indirizzo di pd = << &pd << endl;
   cout << “Valore di pd = << pd << endl;
   cout << “Indirizzo di giorno = “ << &(d.giorno);
   cout << “ size of d.giorno = “ << sizeof(d.giorno) << endl;
   cout << “Indirizzo di mese = “ << &(d.mese);
   cout << “ size of d.mese = “ << sizeof(d.mese) << endl;
   cout << “Indirizzo di anno = “ << &(d.anno);
   cout << “ size of d.anno = “ << sizeof(d.anno) << endl;
}
```


AA 2006/2007

MR

16

Inf Gen II

Strutture e memoria

- Similmente al caso di tipi standard possiamo dichiarare delle variabili di tipo puntatore a struttura, e si può allocare area di memoria per memorizzare strutture.

```
int main () {
 Data *d = new Data;
 d->giorno = 10;
 d->mese = 11;
 d->anno = 2007;
 delete d;
}
```

È un errore non allocare la memoria ed accedere ai campi della struttura.

È un errore non deallocare la memoria precedentemente allocata.

AA 2006/2007

MR

17

Inf Gen II

Accesso agli elementi di una struttura

- Se **ps** è una variabile di tipo un puntatore ad una struttura avente **field** come identificatore di campo, il valore del campo è accessibile in due modi analoghi:
  - **(*ps).field**
  - **ps->field**

  Il secondo modo è il più utilizzato.
- Esempio:

```
#include <iostream>
using namespace std;
struct complex { double re, im; };

int main() {
 complex c;
 complex *pc = &c;
 c.re = 2.5;
 pc->im = 3;
 cout << "Parte reale = " << pc->re << endl;
 cout << "Parte immaginaria = " << c.im << endl;
}
```

AA 2006/2007

MR

18

Inf Gen II

AA 2006/2007

MR

Inizializzazione di una struttura

- Nella dichiarazione di una struttura possiamo introdurre una o più funzioni *con lo stesso nome della struttura*.
- Tali funzioni possono essere usate per *inizializzare* i campi della struttura e quindi una variabile di tipo struttura.
- La funzione di inizializzazione non ha tipo di ritorno ed è detta "***costruttore***".
- Esempio:

```
// Struttura per rappresentare la data
 struct data {
   int giorno;
   int mese;
   int anno;
   data(int g, int m, int a) {
     giorno = g; mese = m; anno = a;
   }
 };

 data D = data(21, 3, 2006); //inizializza i campi di D
```

19

Inf Gen II

AA 2006/2007

MR

Inizializzazione di una struttura

- Il C++ mette a disposizione due modi per dichiarare una variabile di un tipo non standard.
- Ad esempio, se vogliamo dichiarare una variabile dal nome "oggi" di tipo Data come da dichiarazione precedente, possiamo procedere come sotto:
  - data oggi(21, 3, 2006);
  - data oggi = data(21, 3, 2006);

20

Costruttori di una struttura

- Si possono definire **vari costruttori** per una struttura a patto che essi differiscano per il numero e/o il tipo di parametri.
- Il compilatore capisce quale costruttore si intende usare nei vari casi in base al numero e/o al tipo di parametri specificati nella chiamata.

Inf Gen II AA 2006/2007 MR

Costruttori di una struttura

```
struct data
  {  int giorno;
     int mese;
     int anno;
     data(int g){
     giorno=g;}
     data(int g, int m){
     giorno=g; mese=m; anno=2006;}
     data(int g, int m, int a){
     giorno=g; mese=m; anno=a; } };
```

Costruttore a un argomento

Costruttore a due argomenti

Costruttore a tre argomenti

Inf Gen II AA 2006/2007 MR

Costruttori di una struttura

- Quando il costruttore non viene definito, il C++ usa per costruire la struttura un **costruttore di default** che alloca memoria per i vari campi della struttura e lascia indefinito il loro valore.
- Esempio:

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

data oggi;
```

Viene invocato il costruttore senza argomenti predefinito

Costruttori di una struttura

- Appena definiamo un costruttore per la struct, *rinunciamo* ad usare il costruttore di default; il suo uso viene inibito dal compilatore.
  - In sua vece possiamo definire un costruttore senza argomenti.
- Esempio:

```
struct data {
   int giorno;
   int mese;
   int anno;
   data(){giorno=1; mese=1; anno=2000;}
   data(int g, int m, int a){
     giorno=g; mese=m; anno=a;}
};
```

In tal modo è ancora possibile scrivere dichiarazioni del tipo:

```
data oggi;
```

Viene invocato il costruttore senza argomenti, *ma non quello predefinito*, bensì quello definito da noi.

Inf Gen II

AA 2006/2007

MR

Assegnamento e Strutture

- A differenza degli array, l’assegnamento è definita per le variabili di tipo struct.
- L’assegnamento di strutture avviene per valore, e quindi vengono copiati tutti i valori dei membri.
- Esempio:

```
struct complex { double re, im; };
complex c1, c2;
c1.re = 2.5; c1.im = 3;
c2 = c1; //assegnazione di struct
```

25

Inf Gen II

AA 2006/2007

MR

Assegnamento e Strutture

- Per gestire gli array in modo che possano essere copiati, si può incapsulare un tipo array come membro di una struct.
- Esempio:

```
struct int_array { int ia[3]; };
int_array sa, sb;

sa.ia[0]=1; sa.ia[1]=2; sa.ia[2]=3;
sb = sa; //l’array viene copiato!
cout << sb.ia[0] << sb.ia[1] << sb.ia[2] << endl;
```

26

Inf Gen II

AA 2006/2007

MR

Assegnamento e Strutture

- Similmente ai costruttori per l'inizializzazione che abbiamo visto prima, esiste un altro costruttore detto *costruttore per copia.*
  - Il costruttore di inizializzazione è invocato dall'operatore con lo stesso nome della struttura (e.g., data(3,4,2004));
  - Il costruttore per copia è invocato dall'operatore di assegnamento **=**.
- Il **costruttore di copia** è una funzione **automaticamente definita** per ogni struct il cui comportamento è quello di far corrispondere ad uno ad uno i campi delle due struct cui si applica.
- Il suo comportamento è molto **simile** a quello dell'**assegnamento**.
- Il costruttore di copia viene invocato in tre occasioni:
  - **Inizializzazione** di variabile; ad esempio

```
data oggi(27,11,2003); //Costruttore a tre argomenti
data copia_oggi=oggi;  //Costruttore di copia
```

  - **Passaggio** di parametri **per valore;**
  - Quando si usa l'istruzione "**return**".

27

Inf Gen II

AA 2006/2007

MR

Array di strutture

- Dato che una struttura definisce un tipo, possiamo anche definire array di strutture.
- Definita una relazione d'ordine tra due strutture, possiamo adattare tutti gli algoritmi di ordinamento che vedremo in altre lezioni più avanti nel corso, non solo per lavorare su interi, ma anche a lavorare su strutture generiche.

28

Array di strutture

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

int compare(data A, data B) {
   if (A.year < B.year) return 1;
   if ((A.year == B.year) && (A.month < B.month)) return 1;
   if ((A.year == B.year) && (A.month == B.month) && (A.day < B.day)) return 1;
 return 0;
}

void swap(data &A, data &B) {
   data C = A;
   A = B;
   B = C;
}
```

```
// ordinamento di un insieme di date
void bubblesort(data A[], int N) {
    for( int i = 0; i < N – 1; i++)
      for( int j = N - 1; j > i; j--)
        if (compare(A[j], A[j-1]))
           swap(A[j-1], A[j]);
 }
```

Inf Gen II — AA 2006/2007 — MR — 29

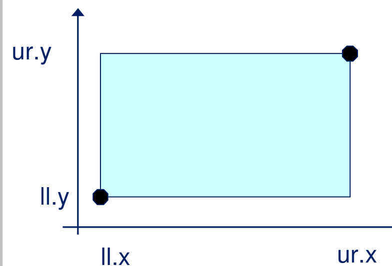
Strutture di Strutture

- È possibile definire strutture di strutture.

```
struct punto {
  double x;
  double y;
}

struct rettangolo {
   punto ll;
   punto ur;
}

double area(rettangolo g) {
    double base = (g.ur.x – g.ll.x);
    double altezza = (g.ur.y – g.ll.y);
    return (base*altezza);
}
```



Inf Gen II — AA 2006/2007 — MR — 30

Inf Gen II

AA 2006/2007

MR

Strutture ricorsive

- La seguente definizione non è lecita:

```
struct S
{ int value;
   S next;  //definizione circolare!
};
```

- La seguente definizione è lecita:

```
struct S
{ int value;
   S *next;
};
```

- Infatti ogni puntatore occupa lo stesso spazio di memoria indipendentemente dal suo tipo.

Inf Gen II

AA 2006/2007

MR

Strutture mutuamente ricorsive

- La seguente definizione non è lecita:

```
struct S1
{ int value;
  S2 *next;  //S2 ancora indefinito
};

struct S2
{ int value;
  S1 *next;
};
```

- `S2` non è stato ancora definito al momento della definizione di `S1`

Inf Gen II

AA 2006/2007

MR

Strutture mutuamente ricorsive

- Dichiarando prima **S2** risulta invece lecita:

```
struct S2;  // dichiarazione di S2

struct S1
{ int value;
  S2 *next; // Ok!
};

struct S2   // definizione di S2
{ int value;
  S1 *next;
};
```

33

Inf Gen II

AA 2006/2007

MR

Definizione di tipi

- In C++ possiamo definire degli identificatvi per riferirsi a tipi definiti dall'utente o tipi primitivi.
- Sintassi:

  typedef <tipo> <new_type_id>;

34

Inf Gen II

AA 2006/2007

Definizione di tipi

- Esempi:

```
typedef struct Point_ Point;
typedef struct Point_ * PointPtr;
typedef char * stringa;

Point P = Point(7,0);
PointPtr Q = new Point(1,9);
stringa s = “Prova”;
```

MR