Laboratorio di Calcolatori 1

Corso di Laurea in Fisica
A.A. 2007/2008

Dott.Davide Di Ruscio

Dipartimento di Informatica

Università degli Studi di L'Aquila

Lezione del 24/04/08

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele

Sommario (II parte)

Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata

- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

McGraw-Hill

Merge di due array ordinati

- L’utente inserisce due array composti da numeri interi ed ordinati in senso crescente
- Le lunghezze dei due array possono essere differenti
- L’elaboratore crea un terzo array ordinato, utilizzando i due array inseriti dall’utente
- Infine, quest’ultimo array viene visualizzato

Pseudocodice

- leggi primo array
- leggi secondo array
- poni a zero i contatori j e k
- for (ripeti un numero di volte pari alla somma delle lunghezze dei due array)
  - if (primo array non terminato AND cella j-esima primo array < cella k-esima secondo vettore OR secondo array terminato)
    - metti nella cella i-esima del terzo array, la cella j-esima del primo array
    - incrementa j
  - else
    - metti nella cella i-esima del terzo array, la cella k-esima del secondo array
    - incrementa k
- visualizza il terzo array

```c
#include <stdio.h>
main()
{
   const int MAX = 100;
   int array1[MAX], array2[MAX], merge[2 * MAX];
   unsigned int n1, n2, im, i1, i2;
   printf ("Lunghezza array 1: ");
   scanf ("%d", &n1);
   printf ("Lunghezza array 2: ");
   scanf ("%u", &n2);
   for (i1 = 0; i1 < n1; i1++)
   {
       printf ("Array1, %d° numero: “,i1+1);
       scanf ("%d", &array1[i1]);
   }
   for (i2 = 0; i2 < n2; i2++)
   {
       printf ("Array2, %d° numero: “,i2+1);
       scanf ("%d", &array2[i2]);
   }
   i1 = 0;
   i2 = 0;
```

```
    for (im = 0; im < n1 + n2; im++)
    {
        if (i1 < n1 && array1[i1] < array2[i2] || i2 >= n2)
        {
            merge[im] = array1[i1];
            i1++;
        }
        else
        {
            merge[im] = array2[i2];
            i2++;
        }
    }
    printf (“Array Finale:\n", merge[im]);
    for (im = 0; im < n1 + n2; im++)
        printf ("%d ", merge[im]);
    printf (“\n");
}
```

Matrice simmetrica

- L'utente inserisce una matrice quadrata, composta da numeri reali, ed il programma verifica (e visualizza) se è simmetrica.
- La dimensione della matrice inserita dall'utente è fissata all'interno del programma
- Ricordiamo che una matrice quadrata è simmetrica sse per ogni *i,j* vale $a_{i,j} = a_{j,I}$
- Le entrate di una matrice simmetrica sono simmetriche rispetto alla diagonale principale (che va dall'angolo in alto a sinistra a quello in basso a destra). Ad esempio:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

Pseudocodice

- for ([contatore riga])
  - for ([contatore colonna])
    - [leggi numero e mettilo nella cella (riga, colonna)]
- [assumi che la matrice sia simmetrica]
- [azzera contatore riga]
- do
  - [contatore colonna ← contatore riga +1]
  - do
    - if ([le celle (riga,colonna) e (colonna,riga) sono diverse])
      - [segnala che la matrice non è simmetrica]
    - [incrementa contatore colonna]
  - while ([contatore colonna è minore della dimensione della matrice AND la matrice è simmetrica])
  - [incrementa contatore riga]
- while ([contatore riga minore di (dimensione della matrice, meno uno) AND la matrice è simmetrica])
- [scrivi il risultato della verifica]

```c
#include <stdio.h>
main()
{
   const unsigned int DIM = 3;
   unsigned int riga, colonna;
   unsigned int simmetrica;
   float matrice [DIM][DIM];
   for (riga = 0; riga < DIM; riga++)
   {
       for (colonna = 0; colonna < DIM; colonna++)
       {
           printf ("numero (%u,%u): ", riga, colonna);
           scanf ("%f", &matrice[riga][colonna]);
       }
   }
   simmetrica = 1;
   riga = 0;
   do
   {
       colonna = riga + 1;
       do
       {
           if (matrice[riga][colonna]!=matrice[colonna][riga])
               simmetrica = 0;
           colonna++;
       } while (colonna < DIM && simmetrica == 1);
       riga++;
   } while (riga < DIM - 1 && simmetrica == 1);
```

```
   if (simmetrica == 1)
        printf ("Matrice simmetrica\n");
   else
        printf ("Matrice non simmetrica\n");
}
```

Calcolo somma di matrici

- Nell'insieme delle matrici quadrate di ordine *n* è possibile definire la somma

- Se

$$A = (a_{ij}),\ B=(b_{ij}) \quad i,j = 1,\ldots,n$$

sono due matrici quadrate di ordine *n*, si chiama matrice somma la matrice

$$C=(c_{ij}) \text{ con } c_{ij}=a_{ij}+b_{ij} \qquad i,j =1,\ldots,n$$

- La matrice somma, ovvero C = A+B, si può calcolare con il seguente programma:

```c
#include <stdio.h>
void main()
{
    const unsigned int DIM = 3;
    unsigned int riga, colonna;
    int a[DIM][DIM], b[DIM][DIM], c[DIM][DIM];
    for (riga = 0; riga < DIM; riga++)
         for (colonna = 0; colonna < DIM; colonna++)
         {
              printf("Inserisci elemento a[%d][%d]",riga,colonna);
              scanf("%d",&a[riga][colonna]);
         }
    for (riga = 0; riga < DIM; riga++)
         for (colonna = 0; colonna < DIM; colonna++)
         {
              printf("Inserisci elemento b[%d][%d]",riga,colonna);
              scanf("%d",&b[riga][colonna]);
         }
    for (riga = 0; riga < DIM; riga++)
         for (colonna = 0; colonna < DIM; colonna++)
             c[riga][colonna]= a[riga][colonna] + b[riga][colonna];

    for (riga = 0; riga < DIM; riga++)
    {
         for (colonna = 0; colonna < DIM; colonna++)
             printf("%d ",c[riga][colonna]);
         printf("\n");
    }
```

Calcolo prodotto di matrici

Se $A \in R^{l \times m}$, $B \in R^{m \times n}$, $C \in R^{l \times n}$, $C = AB$

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} = a_{i1} b_{1j} + \ldots + a_{im} b_{mj}$$

```c
#include <stdio.h>
void main()
{
   const unsigned int DIM_L = 3, DIM_M = 2, DIM_N = 3;
   unsigned int riga, colonna, k;
   int a[DIM_L][DIM_M], b[DIM_M][DIM_N], c[DIM_L][DIM_N];
   for (riga = 0; riga < DIM_L; riga++)
    for (colonna = 0; colonna < DIM_M; colonna++)
    {
        printf("Inserisci elemento a[%d][%d]",riga,colonna);
        scanf("%d",&a[riga][colonna]);
    }

   for (riga = 0; riga < DIM_M; riga++)
    for (colonna = 0; colonna < DIM_N; colonna++)
    {
    printf("Inserisci elemento b[%d][%d]",riga,colonna);
        scanf("%d",&b[riga][colonna]);
    }
```

```
   for (riga = 0; riga < DIM_L; riga++)
     for (colonna = 0; colonna < DIM_N; colonna++)
        {
         c[riga][colonna]=0;
         for (k=0; k < DIM_M; k++)
          c[riga][colonna]=c[riga][colonna]+a[riga][k]*b[k][colonna];
        }
   for (riga = 0; riga < DIM_L; riga++)
   {
        for (colonna = 0; colonna < DIM_N; colonna++)
            printf("%d ",c[riga][colonna]);
        printf("\n");
   }
}
```

Scambio righe

- L’utente inserisce una matrice composta da numeri interi
- Il programma scambia le righe pari con quelle dispari
- Le dimensioni della matrice inserita dall’utente sono fissate all’interno del programma

Pseudocodice

- for ([contatore riga])
  - for ([contatore colonna])
    - [leggi numero e mettilo nella cella (riga, colonna)]

- for ([contatore riga; va da zero al numero i righe della matrice, meno uno; incremento di 2])
  - for ([contatore colonna])
    - [scambia la cella (riga, colonna) con quella (riga + 1, colonna)]
- for ([contatore riga])
  - for ([contatore colonna])
    - [visualizza numero nella cella (riga, colonna)]

```
void main()
{
   const unsigned int DIMRIGA = 4, DIMCOL = 4;
   unsigned int riga, colonna;
   int matrice [DIMRIGA][DIMCOL], temporanea;
   for (riga = 0; riga < DIMRIGA; riga++)
    for (colonna = 0; colonna < DIMCOL; colonna++)
    {
        printf ("numero (%u,%u): ", riga, colonna);
        scanf ("%d", &matrice[riga][colonna]);
    }
   for (riga = 0; riga < DIMRIGA - 1; riga = riga + 2)
    for (colonna = 0; colonna < DIMCOL; colonna++)
    {
        temporanea = matrice[riga][colonna];
        matrice[riga][colonna]=matrice[riga + 1][colonna];
        matrice[riga + 1][colonna] = temporanea;
    }
   for (riga = 0; riga < DIMRIGA; riga++)
   {
    for (colonna = 0; colonna < DIMCOL; colonna++)
        printf ("%d ", matrice[riga][colonna]);
    printf ("\n");
   }
}
```

Tipo di dati strutturati

- Un tipo strutturato non è caratterizzato da un valore semplice, ma da informazione aggregata in diverse componenti.

- Ad esempio un vettore o array consiste in una sequenza di elementi **consecutivi** e **omogenei**:

  - ```
    int matricole_studenti[24];     /*variabile di
                tipo array*/
    ```
  - ```
    Matricole_studenti[4] = 123444; /*assegnamento all'elemento
                    di indice 4*/
    ```

- Il linguaggio C non possiede tipi predefiniti strutturati, tuttavia ha 4 costruttori di tipo:
  - **array**
  - **struct**
  - **pointer**
  - **union**

- Consentono di definire tipi strutturati anche complessi

Il costruttore struct

- Tipo impiegato: nome, cognome, codice fiscale, indirizzo, numero di telefono, eventuali stipendio, data di assunzione e via di seguito.

- Tipo famiglia: un certo insieme di persone, un patrimonio, costituito a sua volta da un insieme di beni, ognuno con un suo valore, un reddito annuo, spese varie, …

- Queste strutture informative sono eterogenee: l'array non si presta a questo tipo di aggregazione.

- Il costruttore di record (parola chiave **struct** in C) è la risposta a questo tipo di esigenze.

Esempi (1)

```
typedef char String[50];

typedef struct    {    int Giorno;
                  int Mese;
        int Anno;
                  }    Data;

typedef struct    {    String   Destinatario;
                       int      Importo;
                       Data     DataEmissione;
                  }    DescrizioneFatture;
```

Esempi (2)

```
typedef enum    {On, Off}       AccType;

typedef struct  {   int             Canale;
                                AccType Accensione;
                    double  CursoreLuminosita, CursoreColore,
    CursoreVolume;
                }   CanaliTV;
```

Esempi (3)

```
typedef enum {Dirigente, Impiegato, Operaio}
    CatType;

typedef struct    {    String        Nome;
                       String        Cognome;
                       int       Stipendio;
                       char          CodiceFiscale[16];
                       Data          DataAssunzione;
                       CatType       Categoria;
                  }    Dipendenti;
```

Dichiarazione di variabili

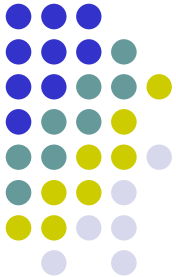
- La dichiarazione di variabili procede poi come al solito:

```
Dipendenti      Dip1, Dip2;
```

- Oppure è possibile definire il nuovo tipo in forma *anonima* e nello stesso tempo dichiarare le variabili:

```
struct {        String      Nome;
                String      Cognome;
                int         Stipendio;
                char        CodiceFiscale[16];
                Data        DataAssunzione;
                CatType             Categoria;
          }     Dip1, Dip2;
```

Accesso alle componenti del record



- Per accedere alle singole componenti del record, si usa una notazione detta *dot notation*:

```
Dip1.Stipendio = Dip1.Stipendio + (Dip1.Stipendio*10) / 100;

Dip1.DataAssunzione.Giorno = 3;
Dip1.DataAssunzione.Mese = 1;
Dip1.DataAssunzione.Anno = 1993;
```

- Se si vuole sapere la prima lettere del cognome di Dip1 è A

```
if (Dip1.Cognome[0] == 'A') ...
```

Accesso alle componenti del record

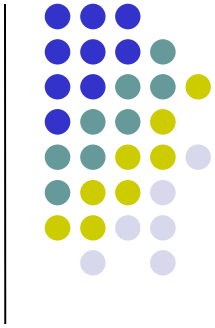
- Si supponga di avere dichiarato una variabile ArchivioFatture

```
DichiarazioneFatture        ArchivioFatture[1000];
```

- Per sapere se la fattura numero 500 è stata emessa entro il 2001 o no, e in caso affermativo, qual è il suo importo, si può scrivere il codice seguente:

```
if   (ArchivioFatture[500].DataEmissione.Anno <= 2000)
     printf("%d", ArchivioFatture[500].Importo);
else
     printf("La fattura in questione è stata emessa dopo il 2000\n");
```

Assegnamento tra record



- Come abbiamo visto, non è permesso scrivere un assegnamento tra array

```
Array2 = Array1;
```

- Invece:

```
Dip1 =  Dip2;
```

  è lecito e fa esattamente ciò che ci si aspetta: copia l'intera struttura `Dip2` in `Dip1`, **comprese le sue componenti che sono costituite da array!**
- Non è invece possibile scrivere una condizione come la seguente:

```
if (Dip1 ==  Dip2)
```

- Il perché di questa stranezza risiede nel modo in cui in C sono realizzati gli array e potrà essere capito tra breve