Lezione 21 e 22

- Allocazione dinamica delle matrici
- Generazione di numeri pseudocasuali
- Funzioni per misurare il tempo
- Parametri del main
- Classificazione delle variabili

Valentina Ciriani (2005-2008)
Laboratorio di programmazione

Valentina Ciriani (2005-2008)
Laboratorio di programmazione

Lezione 21 e 22

Allocazione dinamica delle matrici

Obiettivo:

- Gestire matrici con dimensioni non definite a priori

Allocazione statica

- **Una matrice 3x3 può essere dichiarata come:**
  - `double a[3][3];`
- **Supponiamo di voler progettare una funzione stp che stampi una matrice:**
  - possiamo chiamare la funzione stp sulla matrice a con l'istruzione:
    `stp(a);`
  - la definizione della funzione stampa avrà la forma
    ```
    stp(double a[][3])
    { ... }
    ```
  - che vale solo per matrici 3x3
  - come definire funzioni più generali?

3

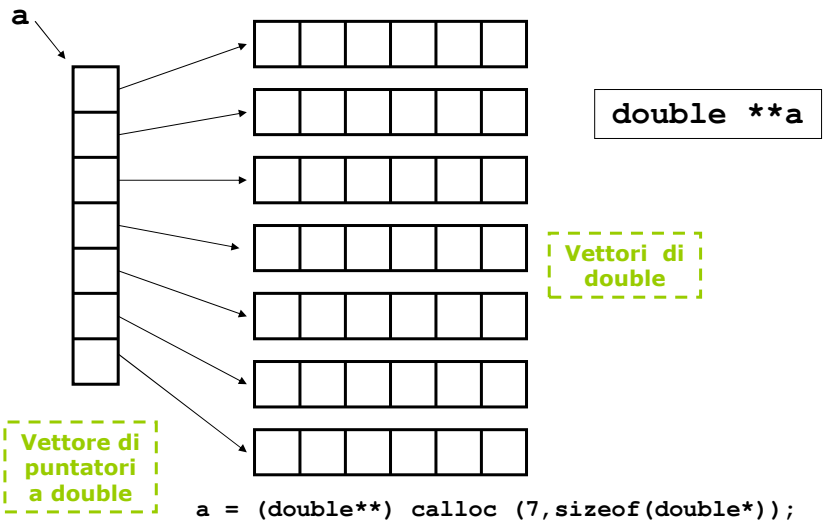
Allocazione dinamica

- **Soluzione: allocare dinamicamente le matrici!**
- **La matrice è quindi:**
  - un array allocato dinamicamente
  - contenete puntatori ad array allocati dinamicamente

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int i,j, n,m;
   double **a; // la matrice e' un puntatore a puntatori a double

   n=4;
   m=5;
   // alloco una matrice con n righe e m colonne
   a= (double**) calloc (n, sizeof(double*));
   for(i=0; i< n; i++)
       a[i]= (double*) calloc (m, sizeof(double));
```

4

Esempio di allocazione




```
a = (double**) calloc (7,sizeof(double*));
for(i=0; i< 7; i++)
   a[i]= (double*) calloc (6,sizeof(double));
```

5

Valentina Ciriani – Università degli Studi di Milano

Utilizzo

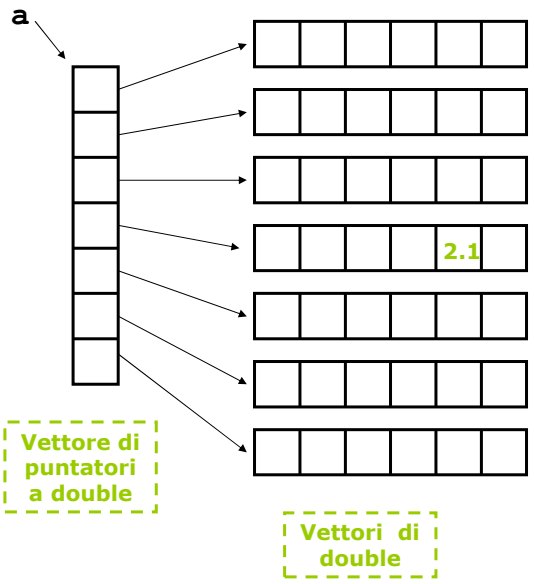
- **Come per i vettori: l'elemento in posizione i, j di una matrice a, allocata dinamicamente, si indica con a[i][j]**

```
// la riempo
for(i=0; i< n; i++)
   for(j=0; j< m; j++)
    a[i][j]= i*j;

// la stampo
for(i=0; i< n; i++)
{
   for(j=0; j< m; j++)
      printf("%5.0f\t",a[i][j]);
   printf("\n");
}
```

6

Valentina Ciriani – Università degli Studi di Milano

Esempio di inserimento



`a[3][4] = 2.1`

7

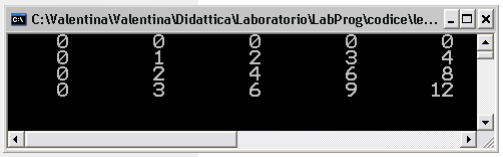
Parametri

- **Possiamo usare ora come parametro di una funzione la matrice allocata dinamicamente**
- **la funzione stampa sarà definita come:**
  ```
  void stp(double** a, int n, int m)
  { ... }
  ```
- **mentre la sua chiamata diventa:**
  ```
  stp(a,n,m);
  ```

8

Esempio

```
#include <stdio.h>
#include <stdlib.h>
void prt (double **a, int n, int m);
int main()
{
   int i,j, n,m;
   double **a;
   n=4;
   m=5;
   a= (double**) calloc (n, sizeof(double*));
   for(i=0; i< n; i++)
       a[i]= (double*) calloc (m, sizeof(double));
   for(i=0; i< n; i++)
      for(j=0; j< m; j++)
       a[i][j]= i*j;
   prt(a,n,m);
   getchar();
   return 0;
}
void prt (double **a, int n, int m)
{
   int i,j;
   for(i=0; i< n; i++){
      for(j=0; j< m; j++)
         printf("%5.0f\t",a[i][j]);
      printf("\n");
   }
}
```



9

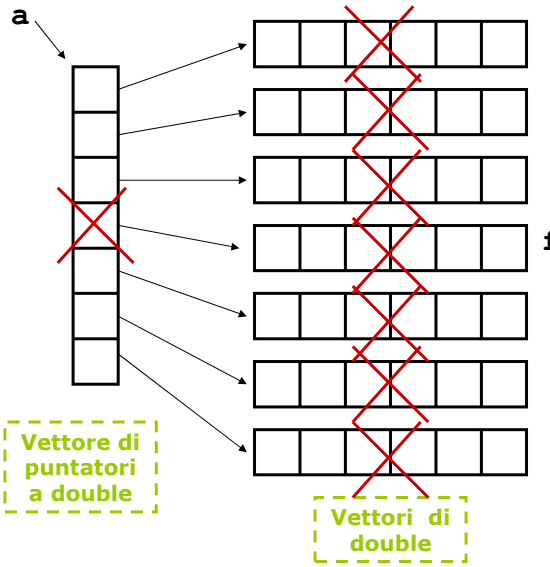
Disallocare una matrice

- **Come tutta la memoria allocata dinamicamente anche le matrici vanno disallocate.**
- **Dobbiamo disallocare prima tutti i vettori che corrispondono alle righe e poi il vettore che abbiamo allocato per primo:**
  - se a è una matrice di n righe e m colone allocata con delle calloc, allora la disalloco con:

```
for (i=0; i<n; i++)
     free(a[i]);
free(a);
```

10

Esempio di disallocazione




```
for (i=0; i<7; i++)
     free(a[i]);
free(a);
```

11

Valentina Ciriani – Università degli Studi di Milano

Valentina Ciriani (2005-2008)
Laboratorio di programmazione

Lezione 21 e 22

L'header <time.h>

Obiettivo:

- Conoscere e saper utilizzate le principali funzioni in time.h

Tempo

- **Per valutare l'efficienza del proprio codice è utile valutare il tempo di esecuzione**
- **Il C ci mette a disposizione alcune funzioni i cui prototipi sono in `time.h`:**
  - `clock_t clock(void)` (`clock_t` è un long)
  - `time_t time(time_t *p)` (`time_t` è un long)
  - `double difftime(time_t t1, time_t t0)`

13

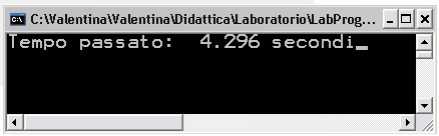
`clock()`

- **La funzione `clock()` restituisce come valore**
  - numero di cicli di clock utilizzato dal programma sino a quel punto
  - che è un tempo relativo (riferito solo al programma in esecuzione)
- **Le unità di clock dipendono dalla macchina**
- **La costante**
  - `CLOCKS_PER_SEC`
  - è definita in `time.h`
  - è usata per trasformare in secondi il valore di `clock()`

14

Esempio

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
   clock_t inizio, fine;
   double diff;
   int i,n;

   inizio = clock();

   for (i=0;i<1000000000;i++)
    n++;

   fine = clock();
   diff = (double) (fine - inizio)/CLOCKS_PER_SEC;

   printf("Tempo passato: \t%.3f secondi",diff);

   fflush(stdin);
   getchar();
   return 0;
}
```



15

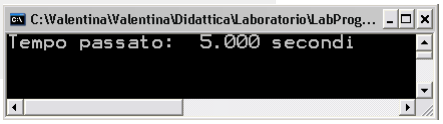
`time()`

- **La funzione `time()` restituisce come valore**
  - il numero di secondi trascorsi dal 01/01/1970
  - che è un tempo assoluto (non è relativo alla singola esecuzione del programma)
- **Se due valori prodotti da `time()` vengono passati a `difftime()`**
  - quest'ultimo restituisce la differenza in secondi.

16

Esempio

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
   time_t inizio, fine;
   double diff;
   int i,n;

   inizio = time(NULL);

   for (i=0;i<1000000000;i++)
    n++;

   fine = time(NULL);
   diff = difftime(fine, inizio);

   printf("Tempo passato: \t%.3f secondi",diff);

   fflush(stdin);
   getchar();
   return 0;
}
```



17

Valentina Ciriani – Università degli Studi di Milano

`clock()` vs `time()`

- **La funzione `clock()` restituisce**
  - un tempo relativo (riferito solo al programma in esecuzione)
- **La funzione `time()` restituisce**
  - un tempo assoluto (non è relativo alla singola esecuzione del programma, ma prende in considerazione il tempo effettivamente trascorso)!
  - se ci sono più programmi in esecuzione **`time()`** conta il tempo complessivo, la funzione **`clock()`** invece calcola il tempo di esecuzione del singolo programma
- **Per avere l'effettivo tempo di calcolo**
  - si usa la funzione **`clock()`**

18

Valentina Ciriani – Università degli Studi di Milano

Valentina Ciriani (2005-2008)
Laboratorio di programmazione

Lezione 21 e 22

Generazione di numeri pseudocasuali

Obiettivo:

- Generare e utilizzare numeri casuali

`rand()`

- **Come esempio di utilizzo di una funzione della libreria standard vediamo la funzione `rand()` (che sta nell'header <stdlib.h>)**

```
int rand(void)
```

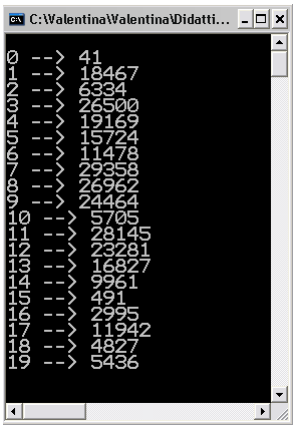
- **Restituisce un intero (pseudocasuale)**
- **Chiamate ripetute generano una sequenza di interi uniformemente distribuiti in [0,RAND_MAX]**
- **RAND_MAX è il maggior numero random che può essere generato (tipicamente vale 32767 = 2^{15}-1)**

20

Inizializzazione

- **Se non viene inizializzato, il generatore di numeri casuali fornisce sempre gli stessi numeri.**

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int i;

   for (i=0;i<20;i++)
      printf("\n%d --> %d", i ,rand());

   fflush(stdin);
   getchar();
   return 0;
}
```



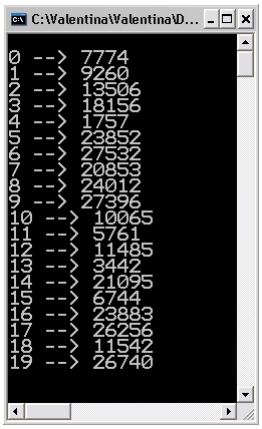
`srand()`

`void srand(unsigned seed)`

- **Inizializza il generatore di numeri random:**
  - in modo che la sequenza generata dalle chiamate di **`rand()`** inizi da una posizione differente
  - all'avvio del programma il generatore si comporta come se fosse stata eseguita **`srand(1)`**
- **Tipicamente viene usato con il parametro time:**
  - **`srand(time(NULL))`**
  - ogni chiamata genera un numero differente!
  - **`time(NULL)`** = numero di secondi dal 01/01/1970

Esempio

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
   int i;
   srand(time(NULL));

   for (i=0;i<20;i++)
      printf("\n%d --> %d", i ,rand());

   fflush(stdin);
   getchar();
   return 0;
}
```



Valentina Ciriani (2005-2008)
Laboratorio di programmazione

Lezione 21 e 22

I parametri del main

Obiettivo:

- Saper usare gli argomenti della riga di comando

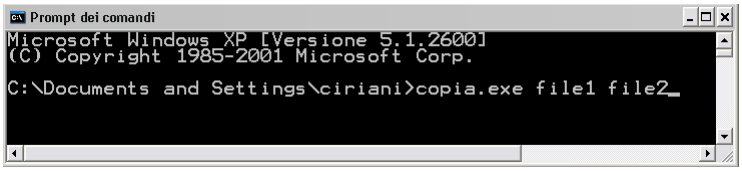
Parametri del `main()`

```
int main (int argc, char *argv[])
```

- **Nel `main()` è possibile utilizzare due parametri:**
  - **`int argc`** (intero)
  - **`char *argv[]`** (vettore di puntatori a char o stringhe)
- **argc contiene il numero elementi sulla riga di comando**
- **l'array argv è**
  - un array di stringhe di dimensione argc
  - contenente la riga di comando

25

Parametri del `main()`



- **Al `main()` di copia.c vengono passati:**
  - argc = 3
  - argv[0] = "copia.exe"
  - argv[1] = "file1"
  - argv[2] = "file2"

26

Esempio copia.c (1)

- **Esempio: scrivere il programma copia.c che**
  - genera una copia esatta di un file
  - il nome del file di partenza e il nome del file di destinazione sono dati come parametri all'eseguibile tramite la riga di comando:

```
>copia.exe file_di_in file_di_out
```

27

Esempio copia.c (2)

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   FILE * file_in;
   FILE * file_out;
   char c;

   // controllo che il numero di argomenti sia corretto
   if (argc !=3) {
      printf("Uso: copia.exe file_di_input file_di_output\n");
      exit(-1);
   }

   // apro il file da copiare
   file_in = fopen(argv[1], "r"); // ricorda che in argv[0] c'e' "copia.exe"!!
   if (file_in == NULL){
      printf("Il file %s non puo' essere aperto\n", argv[1]);
      exit(-1);
   }

   // apro il file in cui copiare il primo
   file_out = fopen(argv[2], "w");
   if (file_out == NULL){
      printf("Il file %s non puo' essere aperto\n", argv[2]);
      exit(-1);
   }
```

28

Esempio copia.c (3)

```
   while(!feof(file_in)){
      fscanf(file_in,"%c", &c);
      fprintf(file_out,"%c", c);
   }


   fclose(file_in);
   fclose(file_out);

   fflush(stdin);
   return 0;
}
```

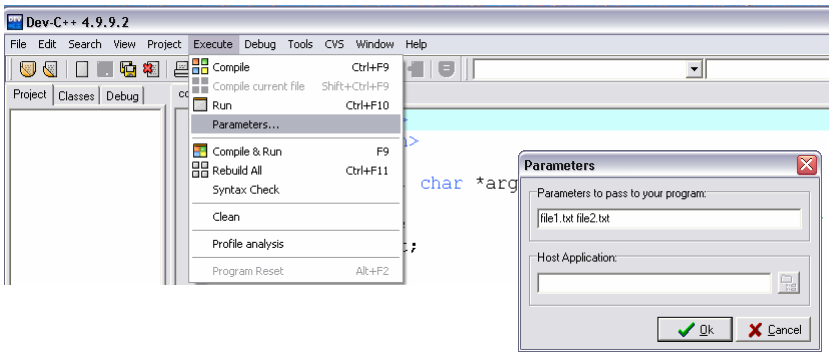
- **Compilo il file copia.c con DevC ma non lo eseguo: mi mancano i parametri!**
- **Eseguo il file copia.exe da riga di comando:**

```
> copia.exe file1.txt file2.txt
```

29

Esempio copia.c (4)

- **Oppure posso inserire i parametri della riga di comando direttamente in DevC:**
  - inserisco i parametri (vedi figura sotto)
  - compilo da DevC
  - eseguo da DevC



30

Valentina Ciriani (2005-2008)
Laboratorio di programmazione

Lezione 21 e 22

Classificazione delle variabili

Obiettivo:

- Conoscere le varie tipologie di variabili che il linguaggio C offre.

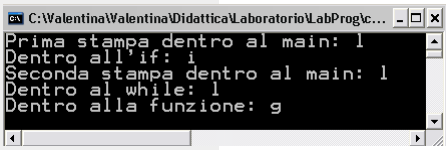
Visibilità

- **Una variabile in C può essere classificata secondo la sua visibilità:**
  - variabile locale che può essere vista solo all'interno del blocco in cui è dichiarata
  - variabile globale che è vista in tutti i punti del codice (non è dichiarata dentro a nessun blocco)
- **Regola di scope:**
  - ogni variabile locale nasce e muore all'interno del blocco in cui viene dichiarata
  - una variabile globale vale sempre
  - ECCEZIONE: se ci sono due variabili con lo stesso nome vale quella del blocco più interno (considerando le globali appartenenti al blocco più esterno)

32

Esempio

```c
#include <stdio.h>
int x;  // variabile globale
char c ='g'; // variabile globale
void funz();
int main()
{
   char c ='l'; // variabile locale
   int y=0; // variabile locale

   // quanto vale c? 'l' o'g'?
   printf("Prima stampa dentro al main: %c\n",c);
   if (y==0){
      char c= 'i';
      // quanto vale c? 'l', 'g' o 'i'?
      printf("Dentro all'if: %c\n",c);
   }
   // quanto vale c? 'l', 'g' o 'i'?
   printf("Seconda stampa dentro al main: %c\n",c);
   while(y==0){
      // quanto vale c? 'l', 'g' o 'i'?
      printf("Dentro al while: %c\n",c);
      y++;
   }
   funz();
   getchar();
   return 0;
}

void funz (){
   // quanto vale c? 'l', 'g' o 'i'?
   printf("Dentro alla funzione: %c\n",c);
}
```



33

Variabili esterne

- **La parola chiave `extern` si usa per informare il compilatore che la definizione del simbolo è presente in un altro file**
- **Esempio**
  - la variabile x è dichiarate in un file "strutture.c"
  - nel file "ordino.c" posso usare la stessa variabile x basta dichiararla come extern:
  - **`extern int x;`**
- **I due file sono compilati separatamente e poi linkati insieme**

34

Variabili globali statiche

- **Se vogliamo fare in modo che una variabile globale sia**
  - globale solo all'interno di un file
  - e quindi non vista da file esterni
  - la dichiariamo come static
  - Es. `static int d;`
- **In questo caso nessun altro file può riferirsi a tale variabile come `extern`**

35

Variabili locali statiche

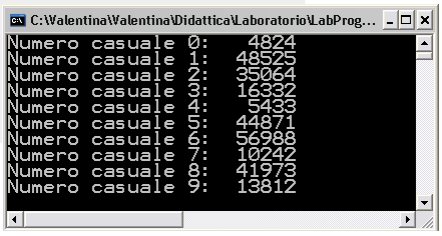
- **Abbiamo appena detto che una variabile locale dichiarata all'interno di un blocco nasce e muore in quel blocco**
- **Possiamo fare in modo che questa variabile mantenga il suo valore anche quando il blocco viene chiuso**
- **Tali variabili sono dette locali statiche:**
  - ES: `static int x;`
- **Può essere utile per esempio all'interno di una funzione.**
  - quando la funzione viene chiamata una seconda volta la variabile mantiene il valore che aveva alla fine della prima chiamata.

36

Esempio variabili locali statiche (1)

```
#include <stdio.h>

unsigned rand (void);

int main()
{
   int i; // variabile locale

   for(i=0; i< 10; i++)
      printf("Numero casuale %d: %6u\n", i, rand());

   getchar();
   return 0;
}


unsigned rand (void){
   static unsigned seme = 5;
   seme = (23478*seme+12888) % 62727;
   return seme;
}
```

L'inizializzazione viene eseguita solo alla prima chiamata

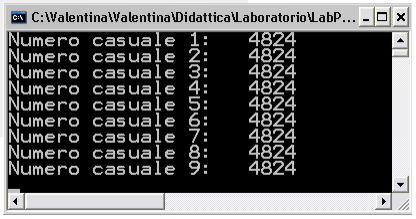


37

Esempio variabili locali statiche (2)

```
#include <stdio.h>

unsigned rand (void);

int main()
{
   int i; // variabile locale

   for(i=0; i< 10; i++)
      printf("Numero casuale %d: %6u\n", i, rand());

   getchar();
   return 0;
}


unsigned rand (void){
  unsigned seme = 5;
   seme = (23478*seme+12888) % 62727;
   return seme;
}
```

La variabile non è più static: viene inizializzata ad ogni chiamata



38

Uso di `static`

- **In C la parola chiave `static` è usata in due modi differenti:**
  - variabili globali: **`static`** fa in modo che la variabile sia locale al file dove è definita e non visibile dagli altri file del programma
  - variabili locali: **`static`** fa in modo che il valore della variabile venga mantenuto all'uscita del blocco in cui è definita e si conservi fino alla fine del programma

39

Altre classi di variabili

- **`auto`, automatic:**
  - variabile locale
  - in disuso: si lascia sottointeso
- **`register`:**
  - dice al compilatore di allocare la variabile direttamente nei registri della CPU
  - per velocizzare in teoria le operazioni sulla variabile
  - Ora in disuso: ci pensa il compilatore a decidere quali variabili sono usate più frequentemente

40