*Programmazione I*

*a.a 2009-2010*

*docente: Carmine Gravino*

# Programmazione modulare
# Librerie e moduli …

**Presentazione realizzata dal Prof. Andrea De Lucia**

# *Progettazione e modularizzazione*

***Progettazione***: l'insieme delle attività relative al concepimento della soluzione informatica di un problema

**dall'architettura, ai dati da manipolare, alle tecniche algoritmiche**

**generalmente si sviluppa a partire da un documento di analisi …**

**… dal cosa al come !**

***Modularizzazione***: dividere per gestire la complessità !

**Unità di programma**

**Alcune già note: funzioni, procedure, ...**

# *Moduli*

- Una unità di programma che mette a disposizione risorse e servizi computazionali (dati, funzioni, …)
- Fondamentale nella realizzazione dei concetti di:
  - **Astrazione**
  - **Information Hiding**
- Riuso di componenti già costruite e verificate
  - **Ad esempio: una volta definite delle funzioni che consentono di risolvere sotto-problemi di utilità generale, come è possibile riusarle nella soluzione di altri problemi ?**

# *Modulo*

- Una unità di programma costituita da:
  - ☞Una Interfaccia
    - **definisce le risorse ed i servizi (astrazioni) messi a disposizione dei “clienti” (programma o altri moduli)**
    - **completamente visibile ai clienti**
  - ☞Una sezione implementativa (body)
    - **implementa le risorse ed i servizi esportati**
    - **completamente occultato**
- Un modulo può usare altri moduli
- Un modulo può essere compilato indipendentemente dal modulo (o programma) che lo usa

**Modulo** *nome*
**Usa** *nomi di moduli*
**Interfaccia**

*dichiarazioni*

**Implementazione**

*dichiarazioni locali*
*definizioni*

**Fine**

***Una visione astratta***

**In C: un opportuno uso di header a source files e delle regole di visibilità ...**
**… convenzioni !**

# *Moduli e C*

- In C non esiste un apposito costrutto per realizzare un modulo; di solito un modulo coincide con un file
- Per esportare le risorse definite in un file (modulo), il C fornisce un particolare tipo di file, chiamato ***header*** file (estensione ***.h***)
  - **Un header file rappresenta l'interfaccia di un modulo verso gli altri moduli**
- Per accedere alle risorse messe a disposizione da un modulo bisogna ***includere*** il suo header file
  - **Concetto già incontrato per le librerie predefinite: ad esempio #include <stdio.h> …**
  - **Per i moduli definiti dall'utente: #include "modulo.h" ...**

# *Moduli e C*

```
Modulo nome
Usa nomi di moduli
Interfaccia

    dichiarazioni

Implementazione

    dichiarazioni locali
    definizioni

Fine
```

**# include**

**Header file: dichiarazioni e prototipi**

**C file:**
**dichiarazioni *static***

**definizioni**

**funz_int.h**

```
// Modulo funz_int: Interfaccia

  int fattoriale(int n);
  int gcd(int a, int b);
  int max(int a, int b);
```

**funz_int.c**

```
// Modulo funz_int: Implementazione

  static int resto(int x, int y);

  int fattoriale(int n) { … }
  int gcd(int a, int b) { … }
  int max(int a, int b) { … }
  static int resto(int x, int y) { … }
```

Cliente

***Cliente: può usare le risorse e i servizi esportati dal modulo***

# *Moduli e librerie di funzioni*

- Il modulo mette a disposizione attraverso la sua interfaccia funzioni e procedure che realizzano la funzionalità
  - il modulo si presenta come una “libreria” di funzioni
  - per l’information hiding
    - nessun effetto collaterale
    - nessuna variabile globale
    - funzioni di servizio nascoste

# *Dichiarazioni static*

- Servono a realizzare l'information hiding ...
- In generale una risorsa (funzioni, dati) è utilizzabile anche se non dichiarata nell'header file
  - **ad esempio, per usare una funzione definita in un altro file, basta dichiararne prima il prototipo**
  - **per usare variabili globali definite in un altro file basta dichiararle come *extern* …**
    - ***esempio*: extern int x;**
- Dichiarazioni ***static*** di funzioni e variabili globali le rendono private al file in cui sono dichiarate
  - **esempi: static int n;**
    **static int resto(int h, int k);**

# *Un esempio ...*

- Ricordiamo il problema: … dati due interi positivi calcolare il massimo e la somma dei rispettivi fattoriali
  - **il modulo funz_int contiene le funzioni max e fattoriale che servono a risolvere il problema**
  - **per la funzione main costruiamo un altro modulo: max_sum**
  - **l'implementazione del fattoriale nel modulo funz_int ...**

```
int fattoriale(int n)
{ int fatt, i;

  if(n < 0)   fatt = -1;     /* condizione d'errore (precondizione) */
  else { fatt = 1;
         for(i =2; i <= n; i++)   fatt = fatt * i;  }
  return(fatt); }
```

# *… il modulo max_sum*

```
#include <stdio.h>
#include “funz_int.h”
main() {
   int n, m, fattn, fattm, massimo, somma;

   do { printf( “Inserisci primo numero positivo: ”);
        scanf(“%d”, &n; } while(n < 0);
   do { printf(“Inserisci secondo numero positivo: ”);
        scanf(“%d”, &m; } while(m < 0);

   fattn = fattoriale(n);
   fattm = fattoriale(m);
   massimo = max(fattn, fattm);
   somma = fattn + fattm;
   printf(“Il massimo dei fattoriali e’: %d \n”, massimo);
   printf( “La somma dei fattoriali e’: %d \n”, somma); }
```

# *Compilazione ...*

- I due moduli possono essere compilati indipendentemente
  - **ad esempio: gcc -c funz_int.c**
    **gcc -c max_sum.c**
- Possibile anche compilarli insieme:
  - **gcc -c funz_int.c max_sum.c**
  - **in entrambi i modi si ottengono i file funz_int.o e max_sum.o**
- Per collegare i due moduli e produrre l'eseguibile:
  - **gcc funz_int.o max_sum.o -o max_sum**
  - **possibile compilazione e collegamento in un sol passo:**
    **gcc funz_int.c max_sum.c -o max_sum**

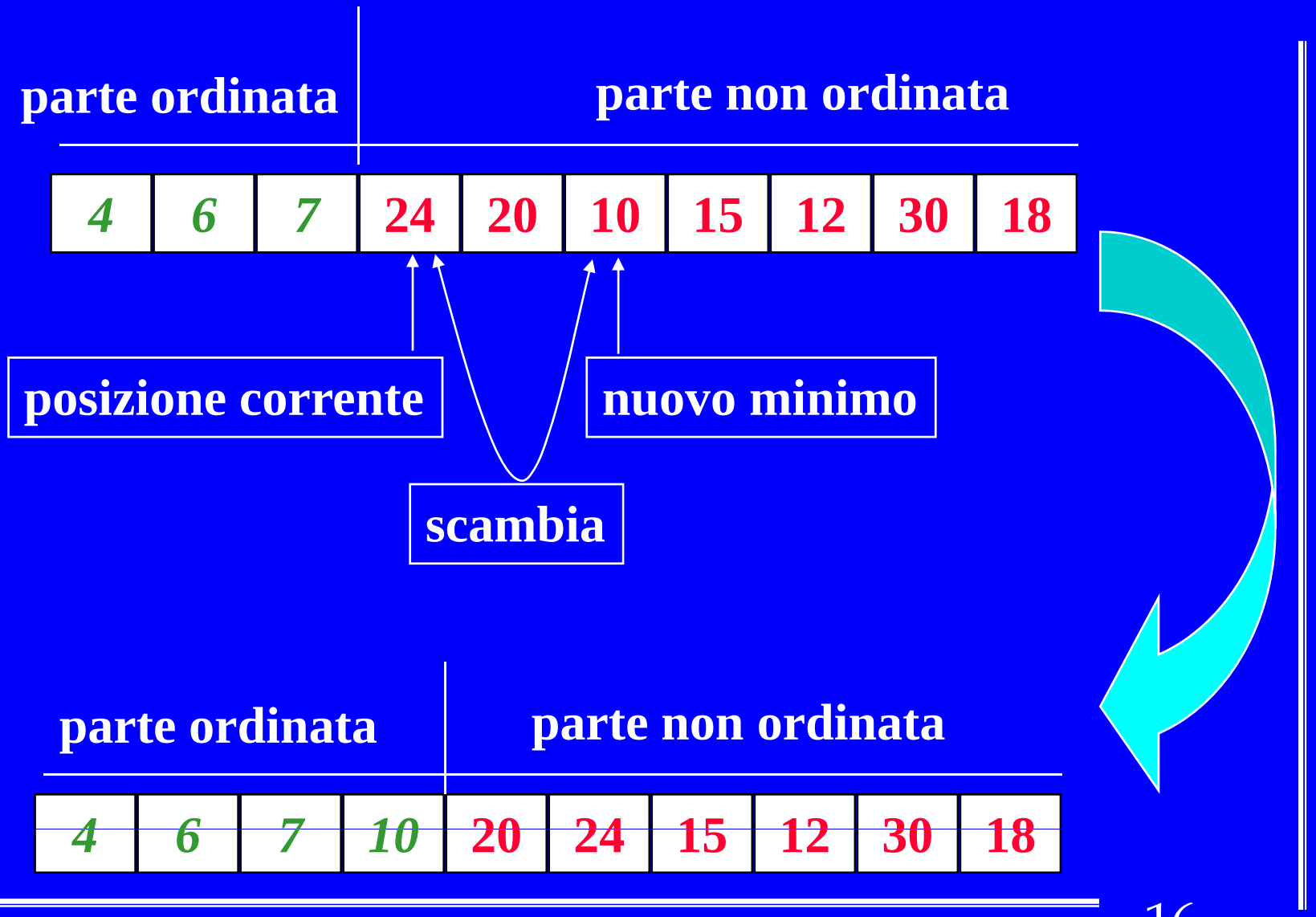
# *Ordinamento di un array*

- Esistono diversi algoritmi di ordinamento iterativi …
  - Selection sort (per minimi successivi)
  - Insertion sort
  - Bubble sort
- … e ricorsivi
  - Quick sort
  - Merge sort
- *Non c'è tempo per vederli tutti …*

# *Selection Sort*

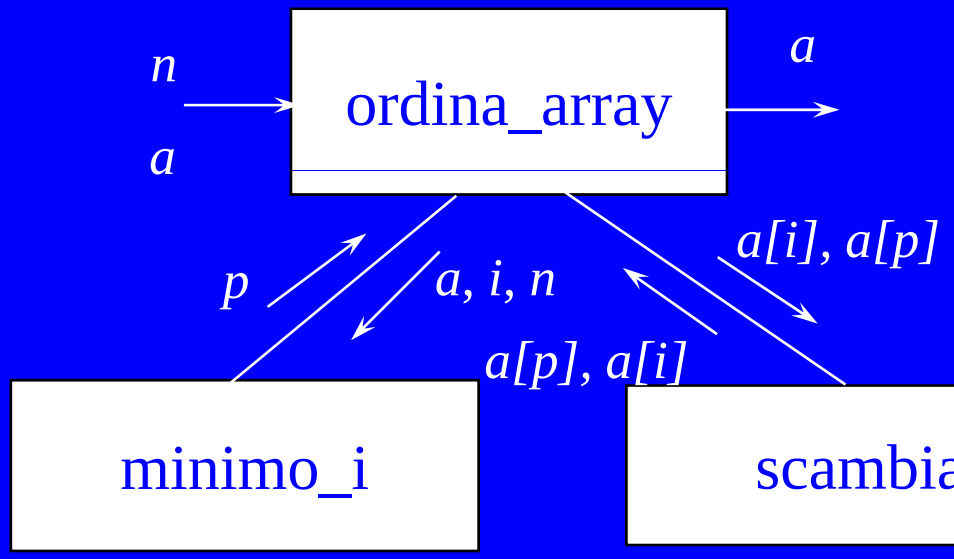
- Effettua una visita totale delle posizioni dell'array
- Per ogni posizione visitata individua l'elemento che dovrebbe occupare quella posizione nell'array ordinato e scambia l'elemento trovato con quello che occupa attualmente la posizione
  - **in questo modo, se *i* è la posizione corrente (0 <= *i* < *n*), tutte gli elementi nelle posizioni comprese tra 0 ed *i*-1 rispettano l'ordinamento;**
  - **quindi l'elemento che deve occupare la posizione *i* sarà il minimo tra quelli nelle posizioni comprese tra *i* ed *n*-1;**
  - **da notare che alla fine l'ultimo elemento (posizione n-1) risulta ordinato ...**

# *Selection Sort*



# *Selection sort: Algoritmo*

- **for(i = 0; i < n-1; i++)**
  - ☞ ***Individua la posizione p dell'elemento minimo compreso tra le posizioni i e n-1 dell'array a***
  - ☞ ***Scambia gli elementi di a di posizioni i e p***




***a è l'array da ordinare ed n il suo riempimento ...***

# *Codice della funzione ordina_array*

```
void ordina_array(int a[], int n)
{ int i, p;
   for (i = 0; i < n-1; i++) {
         p = minimo_i(a, i, n);
         scambia(&a[i], &a[p]);  }
}
```

**minimo_i è dichiarata static nello stesso modulo**

**scambia è definita in un altro modulo (utile.c)**

```
static int minimo_i(int a[], int i, int n)
{ int min, pmin, j;

  min = a[i];  pmin = i;
  for (j = i+1; j < n; j++)
          if (a[j] < min) { min = a[j];   pmin = j; }
  return(pmin); }
```

# *Modulo Vettore*

```
void input_array(int a[], int n);
void output_array(int a[], int n);
int ricerca_array(int a[], int n, int elem);
int minimo_array(int a[], int n);
void ordina_array(int a[], int n);
…
```

**vettore.h**

```
#include <stdio.h>
#include “utile.h” // contiene procedura scambia

static int minimo_i(int a[], int i, int n);
void input_array(int a[], int n) { … }
void output_array(int a[], int n) { … }
int ricerca_array(int a[], int n, int elem) { … }
int minimo_array(int a[], int n) { … }
void ordina_array(int a[], int n) { … }
static int minimo_i(int a[], int i, int n) { … }
…
```

**vettore.c**

# *Codice di alcune funzioni ...*

```
void input_array(int a[], int n)
{   int i;
    for(i = 0; i < n; i++) {
          printf( “Elemento di posizione %d :”, i);
          scanf(“%d”, &a[i]);  } }

void output_array(int a[], int n)
{    int i;
     for(i = 0; i < n; i++)
          printf( “Elemento di posizione %d : %d \n”, i, a[i]);}

int  minimo_array(int a[], int n)
{
    return (minimo_i(a, 0, n));  // restituisce il minimo tra 0 ed n-1
}
```

# *Un programma completo*

```
#include <stdio.h>
#include “vettore.h”
#define NMAX 10
main()
{
    int a[NMAX], n;
    do { printf( “Riempimento array: ”); scanf(“%d”, &n); }
    while(n <= 0 || n > NMAX);

    input_array(a, n);          // Input elementi array
    ordina_array(a, n);         // Ordinamento elementi array
    output_array(a, n);         // Output elementi array ordinati
}
```