

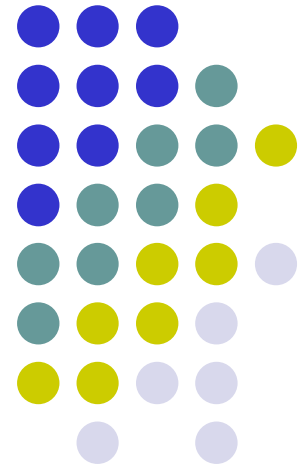
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica

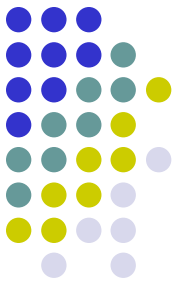
A.A. 2006/2007

Dott. Davide Di Ruscio

Dipartimento di Informatica
Università degli Studi di L'Aquila

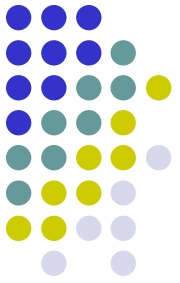


Nota



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

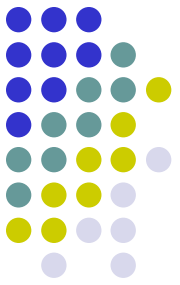
- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
 - tipi
 - definizioni di tipi
 - definizioni di variabili
- Parte esecutiva
 - istruzione di assegnamento
 - istruzioni (funzioni) 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

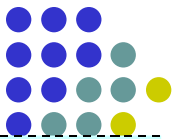
Ceri, Mandrioli, Sbattella
[Informatica arte e mestiere](#)
McGraw-Hill

Visibilità delle dichiarazioni



- Le dichiarazioni locali di una funzione (o del main) sono visibili soltanto all'interno della funzione stessa
- Le dichiarazioni globali sono visibili al main e a tutte le altre procedure e funzioni
- In caso di omonimia le dichiarazioni locali hanno priorità e ricoprono quelle globali

Esempi (entrambi validi)



```
#include <stdio.h>

main()
{
    int fatt(int x);
    int n,k;

    scanf("%d%d",&n,&k);
    bin= fatt(n)/(fatt(k)*fatt(n-k));
    printf("%d",bin);
}

int fatt(int x)
{
    int f,i;

    f=1;
    for (i=2; i<=x; i=i+1)
        f=f*i;
    return(f);
}
```

```
#include <stdio.h>

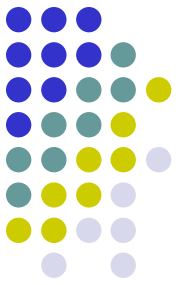
int fatt(int x);

main()
{
    int n,k;

    scanf("%d%d",&n,&k);
    bin= fatt(n)/(fatt(k)*fatt(n-k));
    printf("%d",bin);
}

int fatt(int x)
{
    int f,i;

    f=1;
    for (i=2; i<=x; i=i+1)
        f=f*i;
    return(f);
}
```



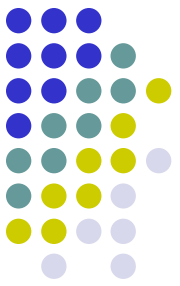
Visibilità delle variabili

- Da quanto sottolineato le variabili locali hanno precedenza come visibilità rispetto alle variabili globali.
 - In una funzione non è visibile una variabile globale avente lo stesso identificatore di una variabile locale.
- I parametri formali delle funzioni sono variabili locali alla funzione.
- Esempio:

```
int x=0;

int g(int a) {
    int x=4;
    return a+x;
}
```

La funzione g non ha visibilità della variabile globale x



Parametri di tipo array

- L'indirizzo di base dell'array viene passato “per valore” alla funzione.

```
typedef double TipoArray[MaxNumElem]
```

- Le tre testate di funzione riportate di seguito sono equivalenti:

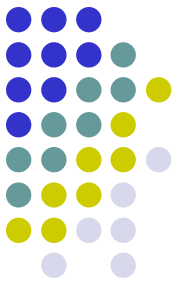
```
/* n è la dimensione dell'array passato */
```

```
double      sum(TipoArray a, int n)
```

```
double      sum(double *a, int n)
```

```
double      sum(double a[ ], int n)
```

NOTA: In C il passaggio dei parametri è **sempre e solo per valore**, tranne per i **vettori** che sono passati **sempre e solo per indirizzo**



```
double mul(double a[], int n)
/* n è la dimensione dell'array
   passato */
{
    int i;
    double ris;

    ris = 1.0;
    for (i=0; i < n; i=i+1)
        ris = ris * a[i];
    return ris;
}
```

- v array di 50 elementi:

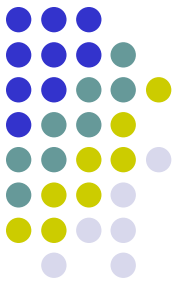
Chiamata *Valore calcolato e restituito*

mul(v, 50) v[0]*v[1]* ... *v[49]

mul(v, 30) v[0]*v[1]* ... *v[29]

mul(&v[5], 7) v[5]*v[6]* ... *v[11]

mul(v+5, 7) v[5]*v[6]* ... *v[11]



Esercizi

Scrivere un programma che utilizzi una funzione per

3. convertire i caratteri minuscoli di una parola in maiuscolo (ricorda che 65->'A', 97->'a')

```
void converti(char *a)
```

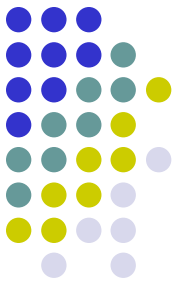
7. restituire la lunghezza di una stringa

```
int strLength(char *a)
```

- confrontare due stringhe; la funzione ritorna 0 se sono uguali, 1 se la prima è maggiore della seconda e -1 se la seconda è maggiore della prima

```
int isEqual(char *a, char *b)
```

Uso interscambiabile di procedure e funzioni



```
intf(int par1)
{
    ...
    return risultato;
}
```

- Essa può essere trasformata nella procedura seguente:

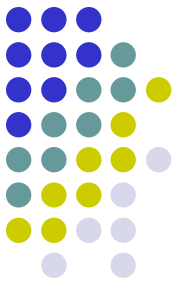
```
void f(int par1, int *par2)
{
    ...
    *par2 = risultato;
}
```

- Successivamente, una chiamata come:

```
y = f(x);
```

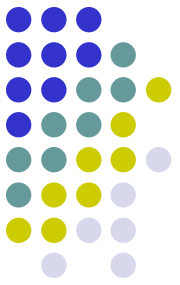
verrà trasformata in:

```
f(x, &y);
```



La standard library del C

- Sottoprogrammi di largo uso predefiniti:
 - Matematica
 - I/O
 - Grafica
- Librerie di sottoprogrammi:
 - Predefinite
 - Costruite dai programmatori applicativi
- Però l'uso di librerie diminuisce la portabilità
- A meno che anche le librerie (almeno le fondamentali) non siano standardizzate
- La grande forza del C: la libreria standard



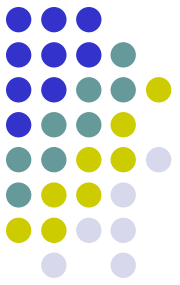
I file header

- Le funzioni della libreria sono disponibili in C come file di codice compilato
- *È compito del programmatore inserire nel programma i prototipi delle funzioni che verranno usate*
- La libreria C comprende alcuni file, chiamati *header file*, che contengono i prototipi di un insieme di funzioni di libreria.

`#include <stdio.h>`

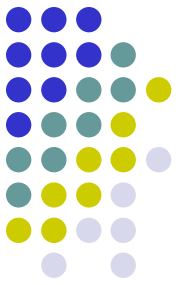
e altre `#include <xxx.h>`

- Il preprocessore copia il contenuto del file `stdio.h` nel programma, inserendo i prototipi delle funzioni che appartengono al gruppo di cui `xxx.h` è il file testata.



Direttive preprocessore

- Quando i file contenenti un programma C vengono compilati, il compilatore, prima di iniziare la compilazione, chiama un **preprocessore** che ha il compito di:
 - Sostituire le costanti o le macro
 - Predisporre il file per una compilazione condizionale
 - Includere in un file altri file

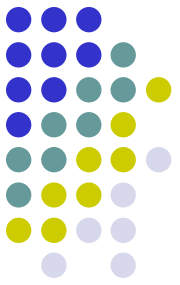


- Le direttive al preprocessore iniziano con il carattere # ed hanno una sintassi indipendente dal C
- Principali direttive al compilatore:
 - #**include** “file_da_includere”
 - #**define** nome_costante valore_costante
 - #**ifdef**, #**ifndef**, #**else**, #**endif** sono utilizzate per la compilazione condizionale

es:

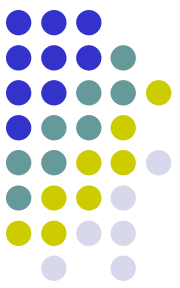
```
#ifndef EOF
#define EOF (-1)
#endif
```

La ricorsione



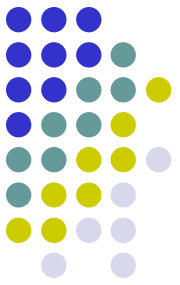
- La formulazione in termini ricorsivi di problemi e algoritmi
- La ricorsione come strumento di programmazione
- L'esecuzione dei sottoprogrammi ricorsivi
- Ulteriori esempi

La formulazione in termini ricorsivi di problemi e algoritmi



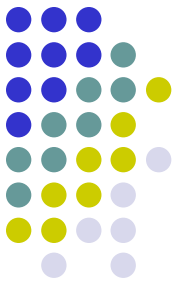
- La ricorsione: che cos'è?
 - Un sottoprogramma P chiama -durante la sua esecuzione- un altro sottoprogramma Q
 - Q a sua volta chiama un terzo R, ...
 - R chiama nuovamente P: (ricorsione indiretta)
 - Oppure P chiama se stesso durante la propria esecuzione (ricorsione diretta)

Paradigma Divide-et-Impera



- Esso consiste nell'eseguire le seguenti tre fasi:
 1. Dividi il problema di partenza in sottoproblemi identici di dimensione minore
 2. Risolvi ricorsivamente, ossia nello stesso modo, i sottoproblemi
 3. Determina e restituisci la soluzione del problema di partenza combinando le soluzioni dei sottoproblemi
- Tale metodologia è convenientemente utilizzata in tutti i quei casi in cui la soluzione di un problema può essere formulata agevolmente in modo ricorsivo

Esempi matematici (1)



- Il fattoriale di un intero non negativo n :

$$\begin{cases} n! = 1 \text{ se } n=0 & \text{(caso base)} \\ n! = n \cdot (n-1)! \text{ se } n > 1 & \text{(caso ricorsivo)} \end{cases}$$

- Esempio:

$$0! = 1$$

$$1! = 1 \cdot 0! = 1 \cdot 1 = 1$$

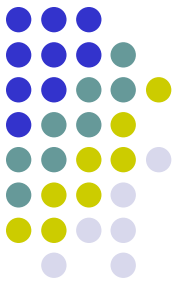
$$2! = 2 \cdot 1! = 2 \cdot 1 = 2$$

$$3! = 3 \cdot 2! = 3 \cdot 2 = 6$$

$$4! = 4 \cdot 3! = 4 \cdot 6 = 24$$

...

Esempi matematici (2)



- I numeri di Fibonacci, $F = \{f_0, \dots, f_n\}$:

$$\left\{ \begin{array}{ll} f_n = 0 \text{ se } n=0 & \text{(caso base)} \\ f_n = 1 \text{ se } n=1 & \text{(caso base)} \\ f_n = f_{n-1} + f_{n-2} \text{ se } n > 1 & \text{(caso ricorsivo)} \end{array} \right.$$

- Esempio:

$$f_0 = 0$$

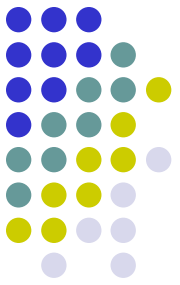
$$f_1 = 1$$

$$f_2 = f_1 + f_0 = 1 + 0 = 1$$

$$f_3 = f_2 + f_1 = 1 + 1 = 2$$

$$f_4 = f_3 + f_2 = 2 + 1 = 3$$

Esempi matematici (3)

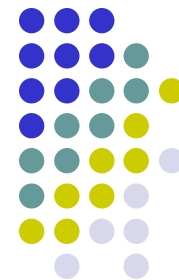


- La sommatoria di una sequenza di numeri

$$\sum_{i=1}^0 a_i = 0$$

$$a_n = a_n + \sum_{i=1}^{n-1} a_i$$

Esempi matematici (4)



- La lista inversa L^{-1} di una lista di elementi $L = \{a_1, \dots, a_n\}$:

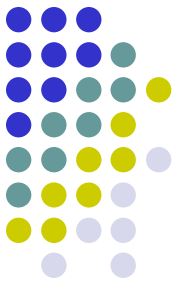
$$\begin{array}{ll} \text{se } n = 1, & L^{-1} = L; \\ \text{altrimenti,} & L^{-1} = \{a_n, (L_{n-1})^{-1}\} \end{array}$$

Dove L_{n-1} indica la lista ottenuta da L cancellando l'ultimo elemento a_n .

- Esempio:

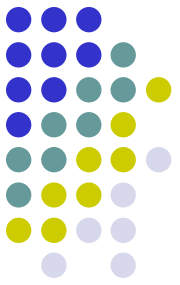
$$\begin{aligned} \{2,7,5,4\}^{-1} &= \{4, \{2,7,5\}^{-1}\} \\ &= \{4,5, \{2,7\}^{-1}\} \\ &= \{4,5,7, \{2\}^{-1}\} \\ &= \{4,5,7,2\} \end{aligned}$$

Generalizzando ...



- In una definizione ricorsiva esistono sempre
 - **Caso base** in cui la soluzione viene specificata in modo diretto
 - **Caso ricorsivo** in cui la soluzione è determinata ricorsivamente, ossia in termini di se stessa applicata su valori o sottoproblemi di dimensione inferiore

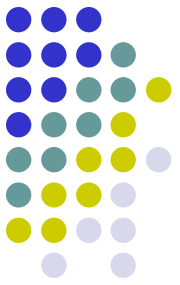
La ricorsione come strumento di programmazione (1)



Calcolo del Fattoriale in modo ricorsivo:

```
int FattRic(int n)
{
    int ris;
    if (n == 0)
        ris = 1;
    else
        ris = n * FattRic(n-1);
    return ris;
}
```

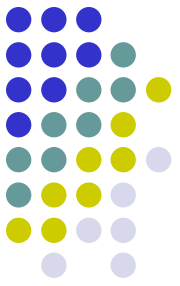
La ricorsione come strumento di programmazione (2)



Calcolo dei numeri di Fibonacci:

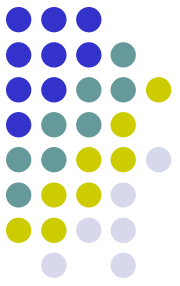
```
int fibonacci(int n)
{
    int ris;
    if (n == 0)
        ris = 0;
    else if (n == 1)
        ris = 1;
    else
        ris = fibonacci(n-1) + fibonacci(n-2);
    return ris;
}
```


L'esecuzione di sottoprogrammi ricorsivi (1)

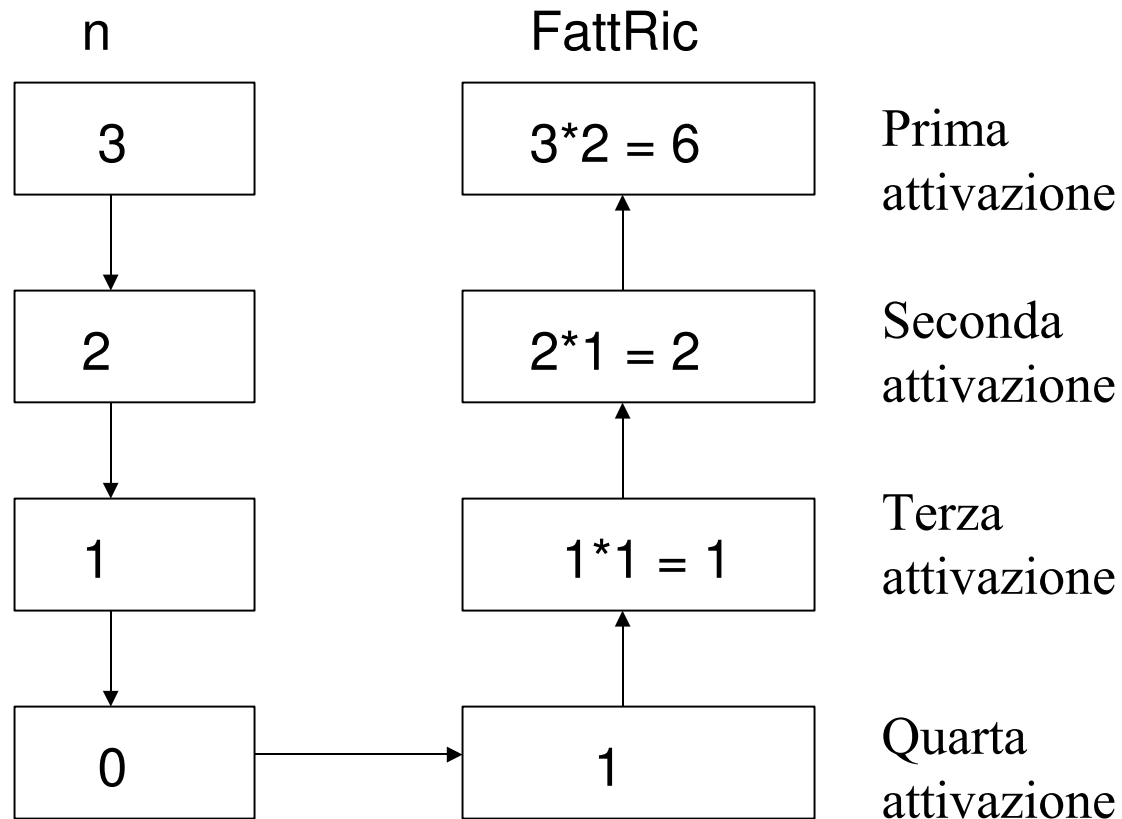


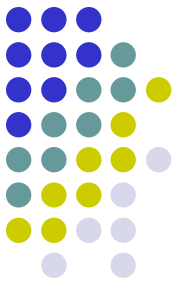
- Calcolo del fattoriale di 3 (secondo lo schema conosciuto):
 - Il *valore* del parametro attuale, 3, viene copiato nel parametro formale, n
 - Ha inizio l'esecuzione di FattRic. Essa giunge a $n * \text{FattRic}(2)$, il cui risultato dovrebbe essere assegnato alla cella ris per poi essere passato al chiamante.
 - A questo punto avviene la nuova chiamata di FattRic.
 - Il nuovo valore del parametro attuale, 2, viene copiato nella cella n, *cancellando il precedente valore 3*

L'esecuzione di sottoprogrammi ricorsivi (2)



record di attivazione





$$6! = 6 * 5! = 720$$

$$5! = 5 * 4! = 120$$

$$4! = 4 * 3! = 24$$

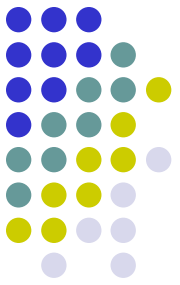
$$3! = 3 * 2! = 6$$

$$2! = 2 * 1! = 2$$

$$1! = 1 * 0! = 1$$

$$0! = 1$$

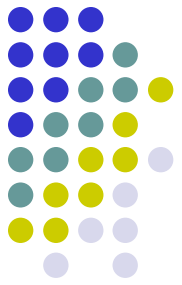
L'esecuzione di sottoprogrammi ricorsivi (3)



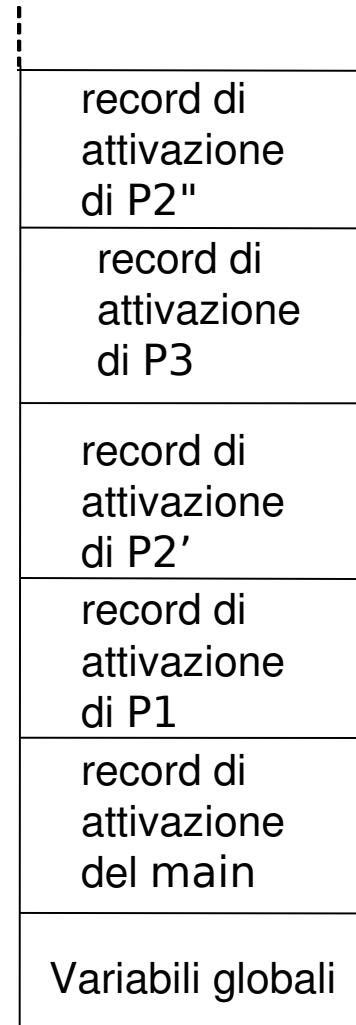
Passaggio parametri -anche- per indirizzo:

```
void incrementa(int *n, int m)
{
    if (m != 0)
    {
        *n = *n + 1;
        incrementa(n, m-1);
    }
}
```

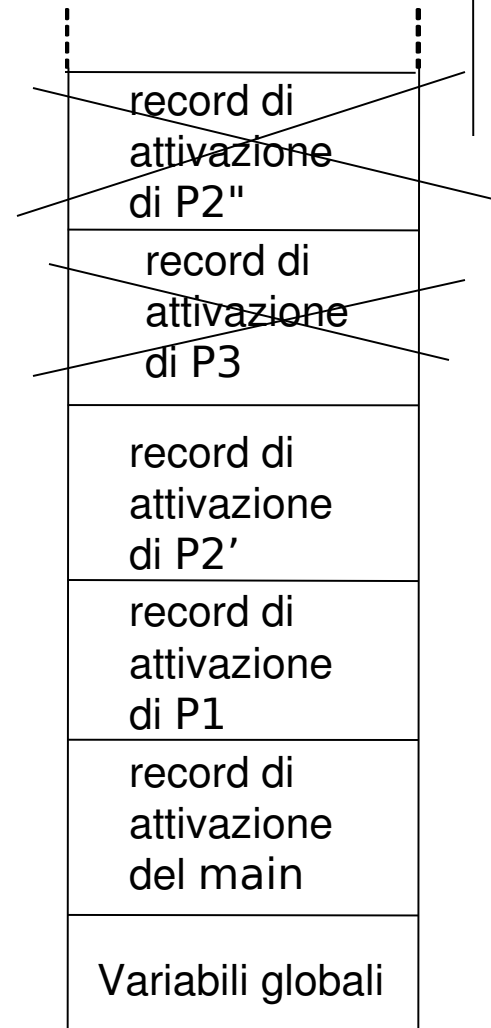
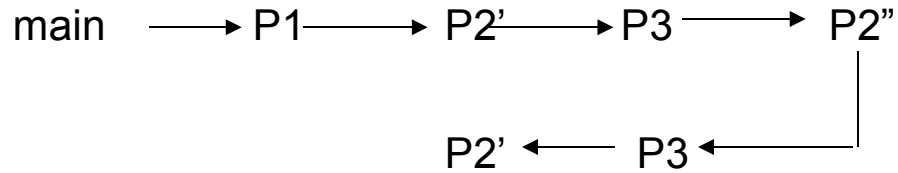
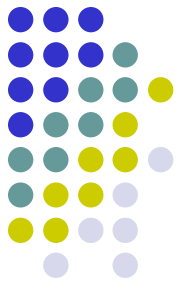
L'esecuzione di sottoprogrammi ricorsivi (3): la gestione a pila della memoria (a)



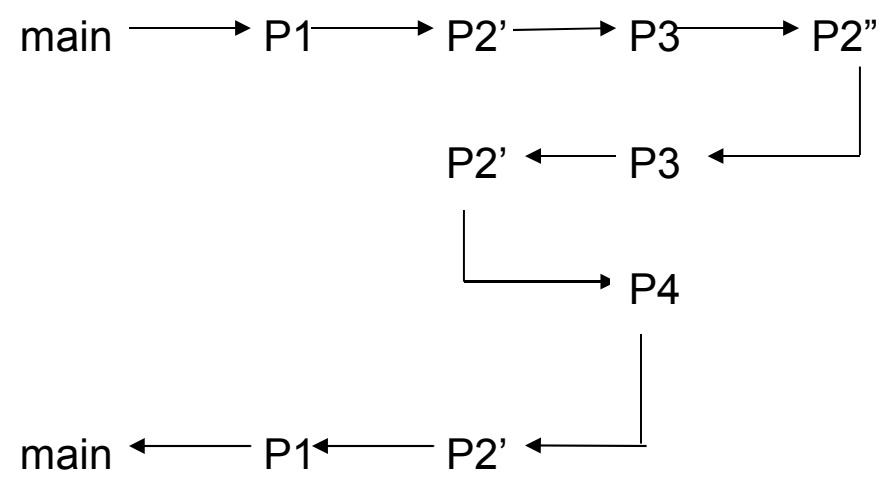
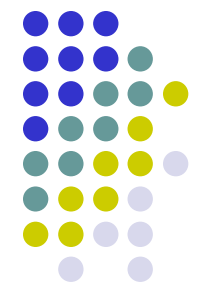
main → P1 → P2' → P3 → P2''

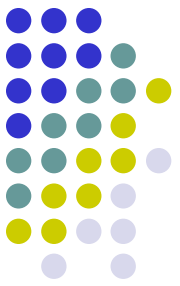


L'esecuzione di sottoprogrammi ricorsivi (4): la gestione a pila della memoria (b)



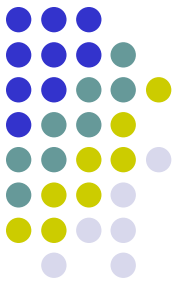
L'esecuzione di sottoprogrammi ricorsivi (5): la gestione a pila della memoria (c)





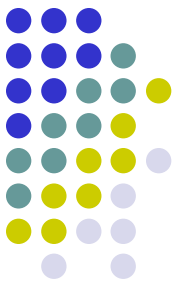
Esercizi su ricorsione:

- Calcolo del massimo comun divisore secondo il metodo di Euclide
- Ricerca elemento in un vettore (sequenziale e binaria)
- Calcolo somma elementi di un vettore
- Stampa elementi di un vettore in ordine inverso
- Massimo elemento di un vettore
- Massimo comun divisore degli elementi di un vettore
- Fusione di due vettori ordinati
- Ordinamento di un vettore
- Inserimento in coda in lista concatenata
- Ricerca elemento in lista concatenata
-



Esercizio MCD

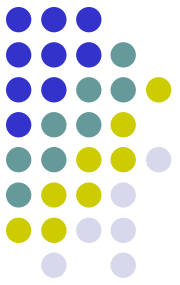
- Il massimo comun divisore (M.C.D.) di due numeri interi, che non siano entrambi uguali a zero, è il numero naturale più grande per il quale possono entrambi essere divisi
- Il massimo comun divisore tra i due numeri a e b viene indicato con $\text{MCD}(a, b)$, o più semplicemente (a, b)
- Due numeri si dicono *coprimi* o *primi tra loro* se il loro massimo comun divisore è uguale a 1. Per esempio, i numeri 9 e 28 sono primi tra loro (ma non sono primi)
- Ad esempio:
 - $\text{MCD}(12, 18) = 6$
 - $\text{MCD}(-4, 14) = 2$
 - $\text{MCD}(5, 0) = 5$.



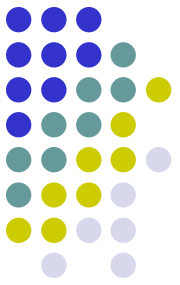
Esercizio MCD: Calcolo

- Il massimo comun divisore può essere calcolato, determinando la scomposizione in fattori primi dei due numeri dati e moltiplicando i fattori comuni, considerati una sola volta con il loro minimo esponente
- Esempio:
 - Per calcolare il $\text{MCD}(18,84)$ si scompongono dapprima i due numeri in fattori primi, ottenendo $18 = 2 \times 3^2$ e $84 = 2^2 \times 3^1$, e poi si considerano i fattori comuni ai due numeri, 2 e 3
 - Entrambi compaiono con esponente minimo uguale a 1, e quindi si ottiene che $\text{MCD}(18,84)=6$

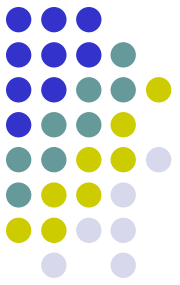
Esercizio MCD: Algoritmo di Euclide



- Calcola il massimo comun divisore MCD di due interi m ed n
 - se $m > n$, $\text{MCD}(m,n) = \text{MCD}(n, m \bmod n)$
 - terminazione: se $n = 0$ ritorna m
- Nell'esempio precedente, si divide 84 per 18 ottenendo un quoziente di 4 e un resto di 12. Poi si divide 18 per 12 ottenendo un quoziente di 1 e un resto di 6. Infine si divide 12 per 6 ottenendo un resto di 0, il che significa che 6 è il massimo comun divisore



```
int MCD (int m, int n)
{
    if (n == 0) return(m);
    return MCD(n, m % n);
}
```



MCD(600,54)

MCD(54, 6)

MCD(6, 0) return 6

MCD(314159,271828)

MCD(271828,42331)

MCD(42331,17842)

MCD(17842,6647)

MCD(6647,4548)

MCD(4548,2099)

MCD(2099,350)

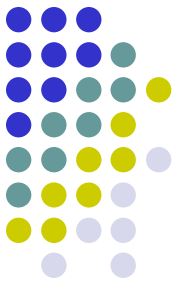
MCD(350,349)

MCD(349,1)

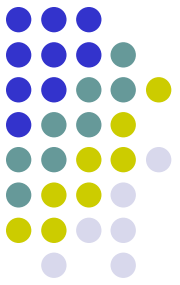
MCD(1,0) return 1

314159 e 271828 sono primi tra di loro

Ricerca binaria in un array



- Dati un array di n interi ordinato in modo crescente ed un intero x forniti in input, determina se l'elemento x è presente nell'array, ossia se esiste una componente dell'array avente lo stesso valore di x
- Idea di risoluzione:
 - Poiché l'array è ordinato, si confronta x con l'elemento centrale
 - Se sono uguali abbiamo trovato l'elemento, altrimenti proseguiamo nel sottovettore a sinistra o a destra dell'elemento centrale a seconda se x è minore o maggiore di tale elemento, dimezzando così lo spazio di ricerca a metà degli elementi.
 - Ripetendo il procedimento giungiamo ad individuare x , se presente, in un numero al più logaritmico in n (base 2) di iterazioni.



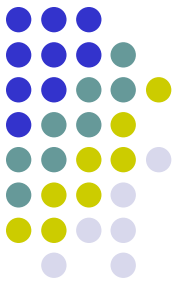
Pseudocodice:

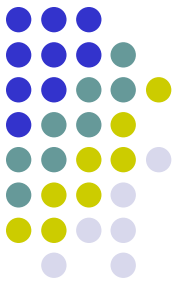
- Inizializza *inf* a 0, *sup* a $n-1$ e *med* alla parte intera inferiore di $(inf+sup)/2$
- Per *i* che varia da 0 a $n-1$ incrementando *i* di 1 ad ogni iterazione
 - Leggi da input il valore da memorizzare nella posizione *i* dell'array a.
- Leggi *x*
- Mentre *inf* è minore o uguale a *sup* e *x* è diverso da $a[med]$
 - Se *x* è maggiore di $a[med]$
 - Poni *inf* uguale a $med+1$
 - Altrimenti
 - Poni *sup* uguale a $med-1$
 - Poni *med* uguale alla parte intera inferiore di $(inf+sup)/2$
- Se *inf* è minore o uguale a *sup*
 - Stampa “L'elemento è presente nell'array in posizione ” *med*
- Altrimenti
 - Stampa “L'elemento non è presente nell'array”

```

#include <stdio.h>          //Versione iterativa
main()
{
    const int n=10;
    int a[n],inf,med,sup,i,x;
    for (i=0; i<n; i=i+1)
        scanf("%d",&a[i]);
    scanf("%d",&x);
    inf=0;
    sup=n-1;
    med=(inf+sup)/2;
    while (inf<=sup && x!=a[med])
    {
        if (x>a[med])
            inf = med+1;
        else
            sup = med-1;
        med=(inf+sup)/2;
    }
    if (inf<=sup)
        printf(" L'elemento e' presente in posizione %d", med);
    else
        printf(" L'elemento non e' presente");
}

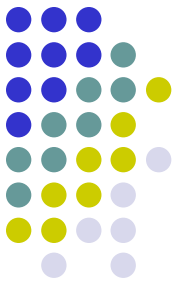
```





```
//Versione ricorsiva
int binSearch(int a[], int sx, int dx, int el)
{
    int x;
    if (dx < sx) return -1;
    x = (dx + sx)/2;
    if (el < a[x]) return binSearch(a,sx,x-1,el);
    else if (el == a[x]) return x;
    else return binsearch(a,x+1,dx,el);
}
```

Ricorsione vs Iterazione



- Ogni programma ricorsivo può anche essere implementato in modo iterativo
- La soluzione migliore (efficienza e chiarezza del codice) dipende dal problema