




**UNIVERSITÀ DEGLI STUDI
DEL SANNIO Benevento**
DING
DIPARTIMENTO DI INGEGNERIA

CORSO DI "PROGRAMMAZIONE I"

Prof. Franco FRATTOLILLO
Dipartimento di Ingegneria
Università degli Studi del Sannio

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 1




Cominciamo con un esempio ...

- ... dati due interi positivi calcolare il massimo e la somma dei rispettivi fattoriali ...

```
Input n, m
if (n >= 0 && m >= 0) {
    Calcola il fattoriale di n e assegnalo ad fn
    Calcola il fattoriale di m e assegnalo ad fm
    Calcola il massimo di fn e fm e assegnalo a max
    somma = fn + fm
    Output max, somma
}
else
    Output messaggio di errore
```


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 2



...finora...

```
#include <stdio.h>
void main() {
    int n, m, fn, fm, max, somma, i;
    scanf("%d %d", &n, &m);
    if (n >= 0 && m >= 0) {
        fn = 1;
        for (i=1; i<=n; i++) fn = fn * i;
        fm = 1;
        for (i=1; i<=m; i++) fm = fm * i;
        max = (fn >= fm) ? fn : fm;
        somma = fn+fm;
        printf("Il massimo e' %d ", max);
        printf("mentre la somma e' %d\n", somma);
    }
    else printf ("Errore \n");
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 3



Utilizzando un sottoprogramma

```
#include <stdio.h>
void main() {
    int n, m, fn, fm, max, somma, i;
    scanf("%d %d", &n, &m);
    if (n >= 0 && m >= 0) {
        fn = fattoriale(n);    fm = fattoriale(m);
        max = (fn >= fm) ? fn : fm;
        somma = fn+fm;
        printf("Il massimo e' %d ", max);
        printf("mentre la somma e' %d\n", somma);
    }
    else
        printf ("Errore \n");
}
```

```
int fattoriale(int n){
    int fatt, i;
    fatt = 1;
    for (i=1; i<=n; i++)
        fatt = fatt*i;
    return fatt;
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 4

Divide et Impera

- I sottoprogrammi sono utilizzati per semplificare la soluzione di problemi complessi attraverso il noto principio del "divide et impera":
 - dividi il problema in sottoproblemi
 - risolvi separatamente i sottoproblemi
 - ricombina i risultati
- Un sottoprogramma consente di raggruppare insieme dati ed azioni in modo da consentirne il trattamento come se si trattasse di azioni primitive
 - un sottoprogramma rappresenta una soluzione ad un sottoproblema
 - progettazione top-down (decomposizione funzionale)

Torniamo all'esempio ...

- ... dati due interi positive, calcolare il massimo e la somma dei rispettivi fattoriali ...

Input n, m

if (n >= 0 && m >= 0) {

Calcola il fattoriale di n e assegnalo ad fn

Calcola il fattoriale di m e assegnalo ad fm

Calcola il massimo di fn e fm e assegnalo a max

somma = fn + fm

Output max, somma }

else Output messaggio di errore

ogni frase in corsivo costituisce un sottoproblema che può essere risolto con un sottoprogramma

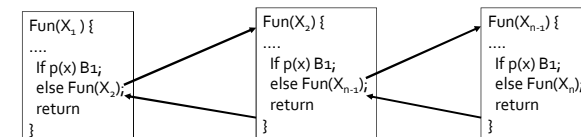
Comunicazione tra sottoprogrammi

- È necessario definire le modalità di passaggio delle informazioni (dati di ingresso) necessarie affinché un sottoprogramma possa risolvere il corrispondente sottoproblema in modo autonomo ed inviare i risultati ottenuti (dati di uscita) al sottoprogramma che lo utilizza
- Es: al sottoprogramma che risolve il sottoproblema "calcola il fattoriale di n ed assegnalo ad fn " bisogna passare il valore n ; il sottoprogramma restituisce il valore da assegnare a fn

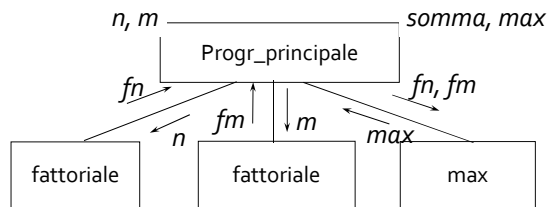
Comunicazione tra sottoprogrammi

- Il meccanismo che consente ad un sottoprogramma di utilizzare un altro sottoprogramma come se fosse una operazione elementare è il meccanismo di chiamata
- Il controllo dell'esecuzione viene trasferito dal sottoprogramma chiamante al sottoprogramma chiamato, insieme ai dati di ingresso (attivazione)
- Al termine dell'esecuzione, il sottoprogramma chiamato restituisce il controllo al sottoprogramma chiamante insieme ai dati di uscita

un sottoprogramma chiamato può a sua volta chiamare un sottoprogramma ...



Decomposizione funzionale: esempio



Struttura dei Sottoprogrammi

- Un sottoprogramma è composto da:
 - una intestazione (interfaccia), che specifica il nome del sottoprogramma e la lista dei parametri (nome, tipo, e indicazione ingresso/uscita) scambiati
 - un corpo, che definisce le azioni e i dati (locali) utilizzati dal sottoprogramma per risolvere il sottoproblema
- La visione di chi utilizza un sottoprogramma è diversa dalla visione di chi lo progetta ed implementa
 - chi utilizza un sottoprogramma è interessato a come è definita la sua interfaccia, non a come è realizzato

Struttura dei Sottoprogrammi

sottoprogramma Nome (lista di parametri) {
 dichiarazione di variabili locali
 istruzioni
 }

Diagram illustrating the structure of a subprogram:

- intestazione** (header) points to the subprogram signature: `sottoprogramma Nome (lista di parametri) {`
- corpo** (body) points to the subprogram body: `dichiarazione di variabili locali` and `istruzioni`

```

int max(int x, int y) {
  int maxVal;
  if (x >= y)
    maxVal = x;
  else
    maxVal = y;
  return maxVal;
}
....
M = max(fn, fm);
  
```

Parametri formali e attuali

- I parametri dichiarati nell'intestazione di un sottoprogramma vengono detti parametri *formali*
- I parametri che compaiono in una chiamata ad un sottoprogramma vengono detti parametri *attuali*
- All'atto della chiamata, i parametri attuali vengono copiati (attualizzati) nei parametri formali del sottoprogramma
 - La corrispondenza tra parametri attuali e parametri formali deve essere in numero, ordine e tipo

Esempio di chiamata di sottoprogramma

```

sottoprogramma Nome (lista di parametri) {
  int a, b, c;
  ....
  Fx(a, b, c);
  ....
}
sottoprogramma Fx(int x, int y, int z)
{
  ....
}

```

Diagramma di annotazione:

- Una freccia punta da `Fx(a, b, c);` a un riquadro con il testo "istruzione di chiamata e parametri attuali".
- Una freccia punta da `int x, int y, int z` a un riquadro con il testo "parametri formali".

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 13

Procedure e Funzioni

- Nella maggior parte dei linguaggi di programmazione esistono due tipi di primitive per realizzare sottoprogrammi:
 - procedura: generalmente usata quando i parametri di output sono più di uno (... o se non ve ne sono)
 - funzione: viene chiamata (e quindi valutata) all'interno di un'espressione e restituisce un valore di ritorno (output) utilizzato nell'espressione
 - di solito, i suoi parametri sono di ingresso ...
 - bisogna dichiarare il tipo del valore di ritorno della funzione
 - in C questa è l'unica primitiva ...

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 14

Sottoprogrammi in C

- In C esiste un'unica primitiva per implementare sottoprogrammi: la funzione ...

Diagramma di annotazione per la funzione `int max(int x, int y)`:

- `int`: tipo di ritorno
- `max`: nome funzione
- `(int x, int y)`: argomenti
- `int max(int x, int y)`: intestazione
- `{`: inizio corpo
- `int z;`: dichiarazione variabile
- `if (x >= y)`: istruzione if
- `z = x;`: istruzione di assegnazione
- `else z = y;`: istruzione di assegnazione
- `return z;`: istruzione return
- `}`: fine corpo

lo scambio dei parametri è per valore

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 15

Definizione di funzione

- L'intestazione di una funzione specifica il tipo di ritorno, il nome e la lista degli argomenti (parametri formali) di una funzione
 - Per ogni argomento viene indicato il tipo e il nome (gli argomenti sono separati da virgole)
- L'istruzione `return` indica il valore (espressione) restituito dalla funzione
 - `return (espressione);`

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 16

Chiamata a funzioni

- Una funzione può essere chiamata in un'altra funzione specificando il nome e la lista di parametri attuali

```
void main( ) {
```

```
    int a, b;
```

```
    printf("a = ");
```

```
    scanf("%d", &a);
```

```
    printf("b = ");
```

```
    scanf("%d", &b);
```

```
    printf("Massimo = %d", max(a, b));
```

```
}
```

chiamata

parametri attuali

Quando si può usare una funzione: Prototipi

- Per poter essere chiamata, una funzione deve essere stata prima dichiarata
 - Il compilatore ha infatti bisogno di controllare il nome e la corrispondenza fra i parametri attuali e i parametri formali
- In realtà, ciò che serve al compilatore è solo l'istestazione della funzione ... non il corpo
 - Il prototipo di una funzione è costituito dalla sua sola istestazione terminata da un punto e virgola
 - Esempio: `int max(int x, int y);` `int max(int, int);`

Definizione e dichiarazione di funzione

- Definizione:** intestazione + corpo
 - Possibile una sola definizione di una funzione
- Dichiarazione:** prototipo
 - Possibili più dichiarazioni della stessa funzione
 - I prototipi possono essere dichiarati localmente al file (all'interno) o globalmente (all'esterno)
- Regole di visibilità:** una funzione f1 è visibile ad un'altra funzione f2 se:
 - il prototipo di f1 è dichiarato localmente al file di f2 e
 - la definizione di f2 segue la definizione o una dichiarazione globale (prototipo) di f1

Esempio completo

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

```
int max(int, int);
```

```
void main( ) {
```

```
    int a, b;
```

```
    printf("a = ");
```

```
    scanf("%d", &a);
```

```
    printf("b = ");
```

```
    scanf("%d", &b);
```

```
    printf("Massimo = %d", max(a, b));
```

```
}
```

```
int max(int x, int y) {
```

```
    int z;
```

```
    if (x >= y)
```

```
        z = x;
```

```
    else
```

```
        z = y;
```

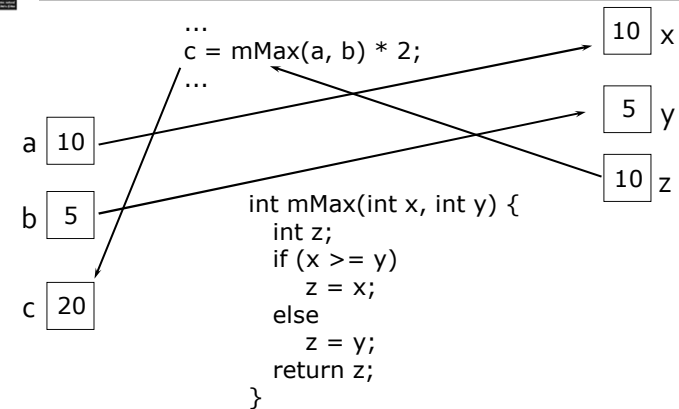
```
    return z;
```

```
}
```

Scambio di parametri per valore

- Per default i parametri di una funzione sono scambiati per valore (... tranne qualche eccezione ...)
- All'atto della chiamata, i valori dei parametri attuali (di input) sono copiati nelle locazioni di memoria dei corrispondenti parametri formali (i parametri attuali possono essere delle espressioni)
 - Eventuali modifiche dei parametri formali non si estendono ai parametri attuali (se questi ultimi sono delle variabili)
 - in pratica, i parametri scambiati per valore sono di ingresso, per cui lo scambio per valore non è sufficiente se sono richiesti più parametri di uscita

Scambio di parametri per valore



Una nota sullo scambio di parametri per valore

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

```
void add3(int n) ;
```

```
void main( ) {
    int n = 5;
    add3(n);
    printf("Il valore di n nel programma principale e': %d\n", n);
}
```

```
void add3(int n) {
    n = n + 3 ;
    printf("Il valore di n nella funzione e' : %d\n", n);
}
```

il valore di n è 5

il valore di n è 8

Puntatori come argomenti di funzioni

- Se l'argomento di una funzione è una variabile puntatore, il passaggio della variabile avviene per riferimento (per indirizzo), ossia la funzione chiamata sarà in grado di modificare il valore della variabile che riceve
- Passaggio argomenti per valore:


```
int numero;
...
square(numero);
```
- Passaggio per indirizzo:


```
SquareR(&numero);
```

Esempio

```
void swap(int x, int y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
}

main() {
    int a = 3;
    int b = 5;
    swap(a, b);
    printf("%d %d\n", a, b);
}
```

- Qual è l'output del programma ?
 - 3 5
- Ciò accade perché la funzione swap agisce solo su una copia delle variabili a e b
- Per far sì che la funzione agisca sulle variabili stesse e non su delle copie, occorre passare gli indirizzi delle variabili

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 25

Esempio

```
void swap(int *px, int *py) {
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}

main() {
    int a = 3; int b = 5;
    swap(&a, &b);
    printf("%d %d\n", a, b);
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 26

Scambio di parametri per riferimento

- Mediante i puntatori è possibile implementare lo scambio dei parametri per riferimento
 - Il parametro attuale e il parametro formale fanno riferimento alla stessa locazione di memoria (viene passato il riferimento alla locazione di memoria)
 - Si dice anche che il parametro formale è un alias del parametro attuale
- Grazie all'impiego dei puntatori le modifiche sui parametri formali si estendono ai parametri attuali
 - In pratica, i parametri scambiati sono sia di ingresso che di uscita

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 27

Scambio di parametri per riferimento

- Il nome del parametro formale nell'intestazione della funzione è preceduto dall'operatore *, e qualunque riferimento al parametro all'interno della funzione è preceduto dallo stesso operatore

```
....
void SommaMedia(double x, double y, double *s, double *m) {
    *s = x + y;
    *m = (*s) / 2;
}
...
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 28

Scambio di parametri per riferimento

- Nell'istruzione di chiamata, i parametri attuali corrispondenti ai parametri formali passati per riferimento sono preceduti dall'operatore di referencing &.

```
void sommaMedia(double x, double y, double *s, double *m) {
    *s = x + y;
    *m = (*s) / 2;
}

void main( ) {
    double a, b, sum, ave;
    ...
    sommaMedia(a, b, &sum, &ave);
    ...
}
```

parametri attuali passati
per riferimento

in C le *procedure*
possono essere
realizzate usando *void*
come tipo di ritorno

Un altro esempio: lo scambio dei valori di due variabili

```
#include <stdio.h>

void scambia(int *, int *);

void main( ) {
    int a, b;
    printf("a = ");
    scanf("%d", &a);
    printf("b = ");
    scanf("%d", &b);
    scambia(&a, &b);
    printf("a = %d, b = %d\n", a, b);
}
```

```
void scambia(int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

parametri di input/output

Puntatori a funzioni

- È possibile definire puntatori a funzioni
 - ciò che contraddistingue una funzione è la coppia di parentesi tonde () : func();
 - la citazione del nome della funzione non seguita dalle parentesi tonde viene interpretata dal compilatore come l'indirizzo della funzione
- Il puntatore ad una funzione può essere manipolato, passato a funzioni ed inserito in un array

```
int abc();
main() {
    printf("l'indirizzo di abc() è %u", abc);
}
```

Uso dei puntatori a funzioni

- È possibile definire una variabile "puntatore a funzione" secondo la seguente sintassi:
 - <return type> <(* name_ptr)> <(arg type, ...)>
- Ad esempio:


```
int (*function_ptr)(char *, float);
```
- indica che `function_ptr` è un puntatore a funzione che ritorna un intero e che riceve due argomenti, rispettivamente di tipo `char*` e `float`
- Poiché un puntatore a funzione è un oggetto che contiene l'indirizzo di una funzione, sarà possibile richiamare una funzione anche attraverso il suo riferimento a puntatore

Esempio

- Una volta conosciuto l'indirizzo di una funzione, la si può invocare mediante il suo indirizzo
 - Esempio:
- ```
void abc() {
 printf("A B C ");
}
main() {
 void (*abcpunt)();
 abcpunt = abc;
 abc();
 (*abcpunt)();
}
```

## Dichiarazione

- La dichiarazione "void (\*abcpunt)( );" significa che abcpunt è una variabile puntatore ad una funzione che ritorna un void
- Con l'assegnazione  
abcpunt = abc;
- si rende effettivo l'uso del puntatore
- **ATTENZIONE:** L'uso delle parentesi è importante, in quanto l'operatore \* ha una precedenza inferiore alla coppia di parentesi ()
- La notazione  
void \*abcpunt( );
- per effetto della precedenza alle parentesi, porta alla definizione di una funzione che restituisce un puntatore a void

## Esempio

- Scrivere un programma "eleva" che riceve, tramite la linea di comando, delle coppie <operazione, intero>, dove operazione specifica il quadrato o il cubo, ed applica l'operazione all'intero; nel caso di errore sull'operazione, il risultato è 0

```
int cubo(int x) {
 return(x*x*x);
}
```


```
int quadrato(int x) {
 return(x*x);
}
```

```
int nulla(int x) {
 return 0;
}
```

## Esempio

```
void main (int argc, char *argv[]) {
 int (*funz)(int); int i = 1;
 while (i<argc) {
 if (!strcmp("quadrato", argv[i]))
 funz = quadrato;
 else if (!strcmp("cubo", argv[i]))
 funz = cubo;
 else
 funz = nulla;
 printf("%d ", (*funz)(atoi(argv[i+1])));
 i += 2;
 }
}
```


```
eleva quadrato 4 cubo 3 qualsiasi 99
16 27 0
```



## La funzione *main*

- In ogni programma C esiste una funzione con nome *main* (funzione principale)
- Tale funzione riceve il controllo dal Sistema Operativo all'atto dell'esecuzione del programma e lo restituisce al termine dell'esecuzione
- La funzione *main* può restituire valori di tipo *int* che possono essere usati per verificare se il programma è terminato correttamente o meno
  - ad es. può essere restituito un codice d'errore
- Per causare la terminazione del programma, è possibile usare la funzione *exit* definita nella libreria *stdlib.h*

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    37




## Funzione *main*: esempi

```
int main() {
 ...
 if(...) return 1; // terminazione anomala
 ...
 return 0 // terminazione normale
}

int main() {
 ...
 if(...) exit(1); // terminazione anomala
 ...
 return 0 // terminazione normale
}
```


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    38



## Variabili (e costanti) locali e globali

- Finora abbiamo assunto che variabili (e costanti) siano dichiarate all'interno di funzioni
  - una tale variabile è detta *locale* ed è visibile solo all'interno della funzione in cui è dichiarata
  - è possibile dichiarare dati anche in blocchi più interni ...
  - tali variabili (costanti) sono dette *automatiche*, perché vengono allocate in memoria a tempo di esecuzione (dell'istruzione dichiarativa) e deallocate al termine del blocco in cui sono dichiarate
- Esistono anche variabili *globali*, dichiarate esternamente alle funzioni ...

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    39



## Variabili e costanti globali

- Una variabile (costante) globale è visibile a tutte le funzioni la cui definizione segue la dichiarazione della variabile nel file sorgente
  - tali variabili (costanti) sono allocate in memoria all'atto del caricamento del programma e la loro deallocazione avviene al termine del programma
- Le variabili globali possono essere usate per scambiare informazioni tra sottoprogrammi ...

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    40

## Scope, Visibilità e Durata

- **Scope**: parte del programma in cui è attiva una dichiarazione
  - definisce quando può essere usato un identificatore
- **Visibilità**: definisce quali variabili sono accessibili in una determinata parte del programma
  - non sempre coincide con lo scope ...
- **Durata**: periodo durante il quale una variabile è allocata in memoria

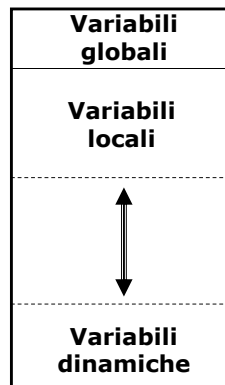
## Scope e Visibilità: un esempio

```
int n;
...
int main() {
 long n;
 ...
 {
 double n;
 ...
 }
 ...
}
```

- int n
  - scope: intero file
  - visibilità: non è visibile nel main
  - durata: tutto il programma
- long n
  - scope: main
  - visibilità: non è visibile nel blocco interno
  - durata: coincide con la durata del main
- double n
  - scope: blocco interno
  - visibilità: coincide con lo scope
  - durata: coincide con la durata del blocco

## Durata: tre aree di allocazione

- **Variabili globali**
  - area di memoria fissata
- **Variabili locali (automatiche)**
  - allocate all'atto dell'esecuzione del blocco in cui sono dichiarate e deallocate alla fine dell'esecuzione del blocco
- **Variabili dinamiche**
  - allocazione e deallocazione dinamica della memoria



## Array come parametri di funzione

- Gli array sono passati sempre per riferimento
  - non è possibile passarli per valore né averli come valori di ritorno di una funzione

```
void Somma(double a[10], double b[10], double c[10], int n)
```

```
void Somma(double a[], double b[], double c[], int n)
• nel primo caso sussiste la precondizione n <= 10 ...
• nel secondo caso possono essere passati array di cardinalità qualsiasi ...
```

## Array come parametri di ingresso

- Con il passaggio di array per riferimento, bisogna stare attenti a non modificare i parametri che si intendono solo di ingresso
  - È possibile il passaggio per costante ...
- Scambio di parametri per costante
  - stesso meccanismo dello scambio per riferimento, ma il parametro non è modificabile
  - utile, oltre che per gli array, per il passaggio di grosse strutture dati di input ...

## Esempio: somma di vettori

```
const int NMAX = 20;

void Somma(const double a[], const double b[], double c[], int n){
 int i;
 for(i = 0; i < n; i++) c[i] = a[i] + b[i];
}

void main() {
 int riemp;
 double x[NMAX], y[NMAX], z[NMAX];

 Somma(x, y, z, riemp); /* deve essere riemp <= NMAX */

}
```

## Visita degli elementi di un array

- Visita totale: vengono analizzati tutti gli elementi
  - In questo caso bisogna usare un ciclo a conteggio
    - È il caso di alcuni esempi precedenti
- Visita finalizzata: la visita termina quando un elemento dell'array verifica una certa condizione
  - In questo caso bisogna usare un ciclo a condizione iniziale
    - due condizioni di uscita: una sull'indice di scansione (visitati tutti gli elementi si esce comunque dal ciclo) e l'altra che dipende dal problema specifico ...
  - Esempio: la ricerca di un elemento termina se è stato trovato l'elemento ...
    - ... in ogni caso, si termina se si sono visitati tutti gli elementi dell'array senza trovare l'elemento dato

## Ricerca di un elemento in un array

```
const int FALSE= 0, TRUE= 1;
int ricerca(const int a[], int n, int elem) {
 int i = 0; /* indice dell'array */
 int trovato = FALSE; /* indica se elem è stato trovato */

 while(i<n && !trovato) /* visita finalizzata */
 if(a[i] == elem)
 trovato = TRUE; /* permette di uscire dal ciclo */
 else i++; /* se non trovato incrementa l'indice */
 if(!trovato) i = -1; /* se non trovato restituisce -1 */
 return i; /* altrimenti la posizione dell'elemento */
}
```

## Ricerca della posizione del minimo

- Visita totale: per trovare il minimo bisogna analizzare tutti gli elementi
- Descrizione informale dell'algoritmo: si assume inizialmente come minimo il primo elemento e se ne memorizza la posizione. Ad ogni passo si confronta l'elemento corrente con il minimo corrente: se il primo è minore, allora questo diventa il nuovo minimo corrente e se ne memorizza la posizione

## Ricerca della posizione del minimo

```
int minimo(const int a[], int n) {
 int i, pmin, min; // min e' il minimo corrente
 // pmin e' la posizione del minimo
 min = a[0]; // inizialmente il minimo è a[0]
 pmin = 0;
 for(i = 1; i < n; i++) // visita totale
 if(a[i] < min) { // trovato il nuovo minimo a[i]
 min = a[i];
 pmin = i;
 }
 return pmin;
}
```

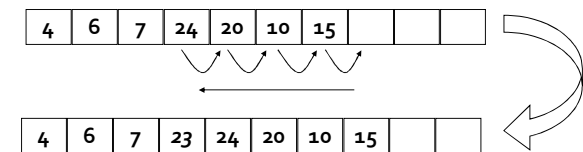
## Inserimento di un elemento in un array

- Dato un array  $a$  di riempimento  $n$  e cardinalità NMAX, inserire un elemento  $el$  in posizione  $pos$ . Sia  $a1$  il nuovo array e  $n1$  il suo riempimento
  - Precondizione:  $0 \leq pos \leq n < NMAX$ 
    - l'array non deve essere "pieno" e l'elemento deve essere inserito in una posizione "valida"
  - Postcondizione:
    - nel nuovo array tutti gli elementi, a partire dalla posizione  $pos$ , sono spostati in avanti di una posizione
    - in posizione  $pos$  si troverà l'elemento  $el$

## Descrizione dell'algoritmo

- l'array  $a$  e il suo riempimento  $n$  sono parametri di I/O
- si spostano in avanti tutti gli elementi dell'array  $a$  compresi tra le posizioni  $pos$  e  $n-1$
- si inserisce  $el$  in posizione  $pos$
- si incrementa  $n$

Esempio: inserire 23 in posizione 3



### Codice della funzione

```
const int FALSE= 0, TRUE= 1;
const int NMAX=100;
void inserisci(int a[], int *pn, int el, int pos, int *err) {
 int i;

 if((*pn < NMAX) && (pos >= 0) && (pos <= *pn)) { // preconditione
 for(i = *pn; i > pos; i--) a[i] = a[i-1]; // shift a destra
 a[pos] = el;
 *pn = *pn + 1;
 *err = FALSE;
 }
 else *err = TRUE;
}
```

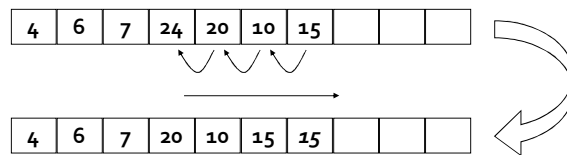
### Eliminazione di un elemento

- Dato un array  $a$  di riempimento  $n$  e cardinalità  $NMAX$ , eliminare l'elemento in posizione  $pos$ . Sia  $a1$  il nuovo array e  $n1$  il suo riempimento
  - Precondizione:  $0 \leq pos < n \leq NMAX$ 
    - l'array non deve essere "vuoto" e la posizione dell'elemento da eliminare deve essere "valida"
  - Postcondizione:
    - nel nuovo array tutti gli elementi a partire dalla posizione  $pos+1$  sono spostati indietro di una posizione

### Descrizione dell'algoritmo

- L'array  $a$  e il suo riempimento  $n$  sono parametri di I/O
- Si spostano indietro tutti gli elementi dell'array  $a$  compresi tra le posizioni  $pos+1$  e  $n-1$
- Si decrementa  $n$

Esempio: eliminare l'elemento in posizione 3



### Codice della funzione

```
const int FALSE= 0, TRUE= 1;
const int NMAX = 100;

void elimina(int a[], int *pn, int pos, int *err) {
 int i;

 if((*pn > 0) && (pos >= 0) && (pos < *pn)) { /* preconditione */
 for(i = pos; i < *pn-1; i++) a[i] = a[i+1]; /* shift a sinistra */
 *pn = *pn - 1;
 *err = FALSE;
 }
 else *err = TRUE;
}
```

### Errore tipico

- Un errore tipico consiste nell'eseguire il ciclo a conteggio scandendo le posizioni in senso decrescente
- Il risultato è la ricopia a sinistra dell'elemento che si trova in posizione finale

The diagram illustrates a common mistake in shifting elements in an array. It shows three states of an array of size 7:

- Initial array: [4, 6, 7, 24, 20, 10, 15]
- After shifting elements from index 5 to 4: [4, 6, 7, 24, 20, 15, 15]
- Final array after shifting: [4, 6, 7, 24, 15, 15, 15]

Arrows indicate the movement of elements: from index 5 to 4, and from index 6 to 5.

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 57

### Ricerca lineare in un array ordinato

- Se l'array è ordinato in senso crescente, non è necessario arrivare alla fine dell'array per stabilire che l'elemento non è stato trovato ...
- ... ci si può fermare appena si trova un elemento maggiore (o uguale) a quello dato ...
  - maggiore --> non trovato !
  - uguale --> trovato !
- Ovviamente, se l'elemento è maggiore di tutti quelli presenti nell'array, allora si visiterà l'intero array (caso peggiore) ...

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 58

### Esempio

- Dato l'array ...

The diagram shows an array of 10 elements: [4, 6, 7, 10, 12, 15, 18, 20, 24, 30]. An arrow points from the first element to the last, indicating a linear search process.

- La ricerca di 13 e la ricerca di 15 terminano quando l'elemento corrente è 15
- La ricerca di 40 termina quando si sono visitati tutti gli elementi dell'array

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 59

### Ricerca lineare in un array ordinato

```
const int FALSE= 0, TRUE= 1;
int ricercaord(const int a[], int n, int elem) {
 int i = 0; int trovato = FALSE;
 while(i<n && !trovato) /* visita finalizzata */
 if(a[i] >= elem)
 trovato = TRUE; /* permette di uscire dal ciclo */
 else
 i++;
 if(!trovato) /* raggiunta la fine dell'array */
 i = -1;
 else if (a[i] > elem) /* trovato un elemento > n */
 i = -1;
 return i;
}
```

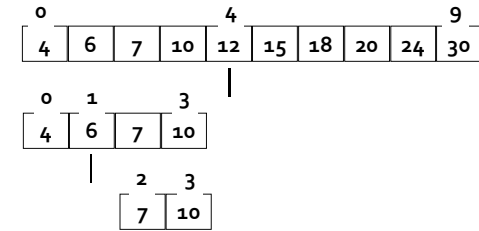
Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 60

## Ricerca binaria (logaritmica) in un array ordinato

- Una strategia migliore consiste nel dividere l'array in due metà e confrontare l'elemento da cercare con l'elemento centrale dell'array
  - uguali --> trovato ... e ci si ferma
  - elemento dell'array maggiore --> continuare la ricerca nella prima metà dell'array
  - elemento dell'array minore --> continuare la ricerca nella seconda metà dell'array
- Se l'elemento non è presente, l'array si ridurrà ad un solo elemento, non divisibile in due (terminazione)
  - nel caso peggiore si visitano  $\log_2 n$  elementi dell'array ...

## Esempio

- Cercare l'elemento 7 nell'array ...



trovato in posizione 2

... e se invece si fosse cercato 8 ? ...

## Ricerca binaria

```
int ricercabin(const int a[], int n, int elem) {
 int h, k, p; int trovato = FALSE;
 h=0; k=n-1; // estremi dell'intervallo in cui ricercare
 while(h<=k && !trovato) {
 p = (h + k) / 2; // posizione centrale
 if(a[p] == elem)
 trovato = TRUE; // permette di uscire dal ciclo
 else if(a[p] > elem)
 k = p-1; // la ricerca continua nella prima metà
 else h = p+1; // la ricerca continua nella seconda metà
 }
 if(!trovato) p = -1;
 return p;
}
```

## Bubble Sort (per scambi)

- Algoritmo iterativo
  - finché l'array non risulta ordinato, si effettua una visita dell'array durante la quale si scambiano gli elementi adiacenti che non risultano ordinati
  - se è stato effettuato almeno uno scambio, allora l'array non è ordinato
  - nota che, ad ogni passo, l'elemento più grande viene portato nella sua posizione finale, per cui, dopo il passo  $i$ -esimo, gli elementi tra le posizioni  $n-i$  ed  $n-1$  risultano ordinati e nelle loro posizioni finali
  - l'algoritmo converge in al più  $n-1$  passi, dove  $n$  è il riempimento dell'array



### Bubble Sort

**1ª iterazione**

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| 10 | 20 | 7  | 18 | 6  | 4  |
| 10 | 7  | 20 | 18 | 6  | 4  |
| 10 | 7  | 18 | 20 | 6  | 4  |
| 10 | 7  | 18 | 6  | 20 | 4  |
| 10 | 7  | 18 | 6  | 4  | 20 |

**2ª iterazione**

|     |    |    |    |    |    |
|-----|----|----|----|----|----|
| 10  | 7  | 18 | 6  | 4  | 20 |
| 7   | 10 | 18 | 6  | 4  | 20 |
| 7   | 10 | 6  | 18 | 4  | 20 |
| 7   | 10 | 6  | 4  | 18 | 20 |
| ... |    |    |    |    |    |

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 65

### Algoritmo di Bubble Sort

- ordinato = FALSO;
- i = 1;
- while((i < n) && (!ordinato))
  - ordinato = TRUE;
  - scambia gli elementi adiacenti che non risultano ordinati tra le posizioni 0 e n-i e poni ordinato a FALSE se viene effettuato almeno uno scambio
- i = i + 1;

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 66

### Codice della funzione ordina\_array

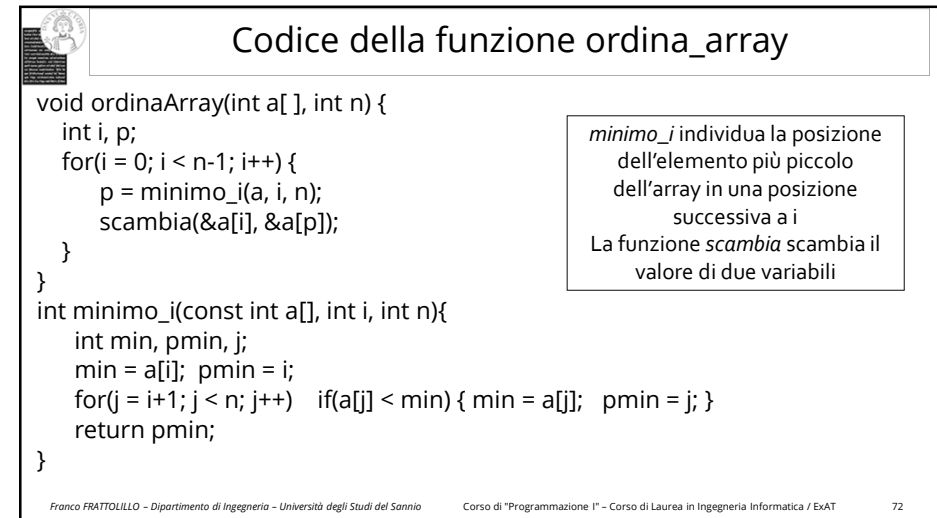
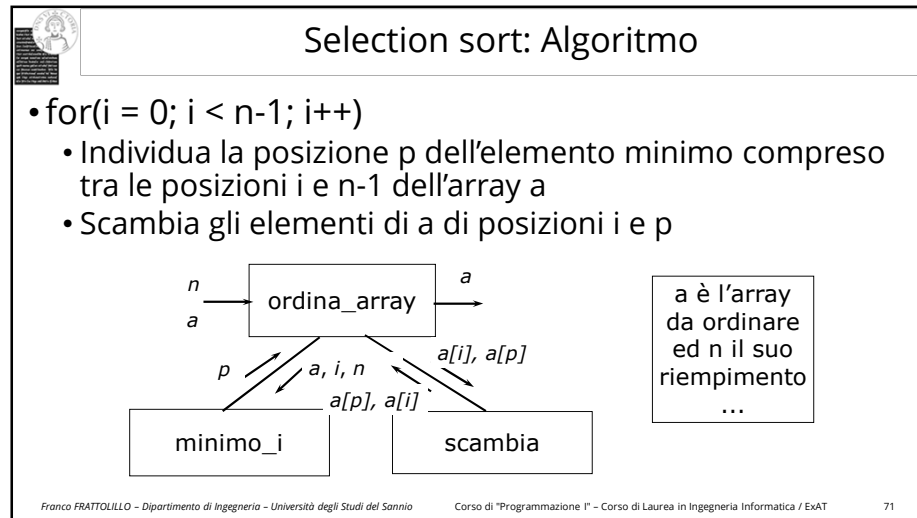
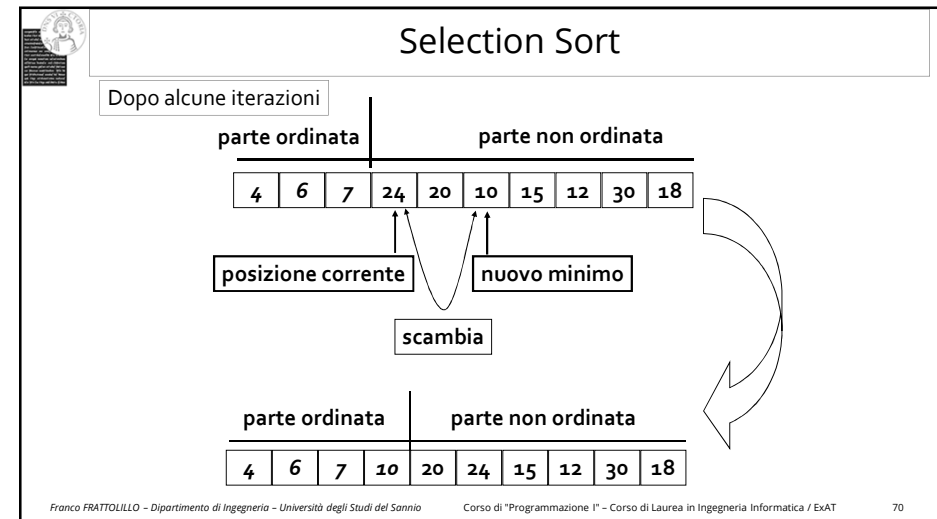
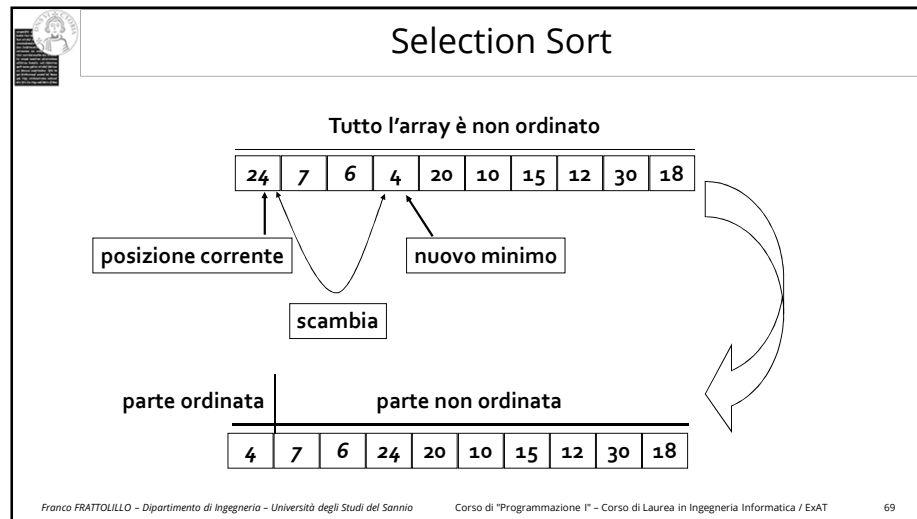
```
void ordinaArray(int a[], int n) {
 int i, j, ordinato;
 ordinato = FALSE;
 i = 1;
 while(i < n && !ordinato) {
 ordinato = TRUE; // assunto ordinato fino a prova contraria
 for(j = 0; j < n-i; j++)
 if(a[j] > a[j+1]) { // coppia adiacente non ordinata
 ordinato = FALSE;
 scambia(&a[j], &a[j+1]);
 }
 i++;
 }
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 67

### Selection Sort (per minimi successivi)

- 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 ( $0 \leq i < n$ ), tutti 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 ...

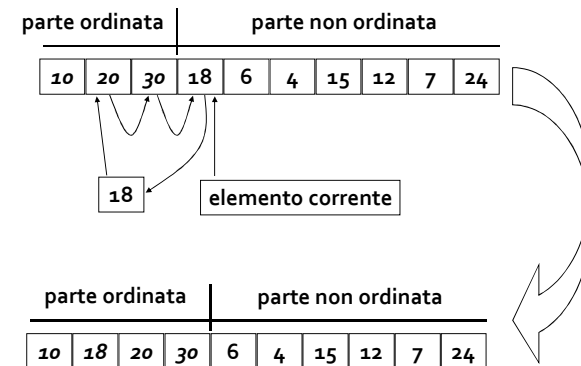
Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 68



## Insertion Sort

- Visita totale dell'array: ad ogni passo gli elementi che precedono l'elemento corrente sono ordinati
  - si inserisce l'elemento corrente nella posizione che garantisce il mantenimento dell'ordinamento
  - gli elementi precedenti maggiori sono spostati in avanti
  - ... il primo elemento è già ordinato ...

## Insertion Sort



## Algoritmo di Insertion Sort

- for( $i = 1$ ;  $i < n$ ;  $i++$ )
  - memorizza  $a[i]$  in una variabile temporanea prossimo
  - sia  $j$  la posizione in cui deve essere inserito prossimo (nota che se  $0 < j < i$ , allora deve essere  $a[j-1] \leq a[j]$ ; se  $j = 0$  allora deve essere  $next < a[j]$ ): sposta in avanti gli elementi di posizioni tra  $i-1$  e  $j$
  - inserisci prossimo in posizione  $j$

## Codice della funzione ordina\_array

```
void ordinaArray(int a[], int n) {
 int i, j, prossimo, terminato;
 for(i = 1; i < n; i++) { // il primo elemento è già ordinato
 prossimo = a[i]; // salva l'elemento corrente
 j = i-1; // indice del ciclo interno
 terminato = FALSE; // ciclo interno: visita finalizzata
 while(j >= 0 && !terminato)
 if(prossimo >= a[j]) // condizione di uscita
 terminato = TRUE;
 else {
 a[j+1] = a[j]; j--; // shift in avanti elemento
 }
 a[j+1] = prossimo; // inserimento elemento corrente in posizione j+1
 }
}
```

## Puntatori e stringhe di caratteri

- Spesso vengono usati i puntatori a caratteri in luogo degli array di caratteri (stringhe), perché il C non fornisce il tipo predefinito stringa
- Esiste una differenza sostanziale tra array di caratteri e puntatori a carattere. Ad esempio in:  
`char *ptr = "Salve mondo";`
- il compilatore non crea una copia della stringa costante "Salve mondo", ma semplicemente crea un puntatore ad una locazione di memoria in cui risiede il primo carattere della stringa costante
- Ciò significa che posso utilizzare il puntatore in modo che punti a qualcosa di diverso senza modificare il contenuto della stringa costante

## Inizializzazione di array e puntatori a caratteri

- Esempio di inizializzazione di array di caratteri:
  - `char caratteri[4] = { 'a', 'A', 'H', 'k' };`
- Esempio di inizializzazione di stringa di caratteri:
  - `char stringa1[ ] = "MMMM";`
  - è aggiunto il terminatore '\0'
- Ecco alcuni esempi di inizializzazione di puntatori a caratteri:
  - `char c = 'a';`
  - `char *carattere;`
  - `carattere = &.....; *carattere = c;`
  - `char *stringa2 = "MMMM";`

## Esempio

```
/* Codice della funzione main */
void main() {
 const int N = 100;
 char linea[N]; int numero;
 printf("Inserisci una linea di testo\n");
 LeggiLinea(linea, N);
 ContaDigit(linea, &numero);
 printf("La linea %s contiene %d numerici", linea, numero);
}
```

## Esempio

```
/* Legge una linea in input */

void LeggiLinea(char *s, int max) {
 char c; int i = 0;
 while((i < max-1) && (c = getchar()) != '\n') {
 (*s) = c; s++; i++;
 }
 *s = '\0';
}
```

## Esempio

```
/*Conta il numero di cifre numeriche*/

void ContaDigit(char *s, int *times) {
 *times = 0;
 for(; *s != '\0' ; s++)
 if ((*s >= '0') && (*s <= '9'))
 (*times)++;
}
```

## Esempi

```
/* Versione con i puntatori della strlen */
int strlen(char *s) {
 int n;
 for(n = 0 ; *s != '\0' ; s++) n++;
 return n;
}

/* Versione con i puntatori della cpy */
void cpy(char *target, char *source) {
 while((*target++ = *source++) != '\0');
}
```

## Funzioni di manipolazione di stringhe


- Alcune funzioni importanti per la manipolazione di stringhe:
  - strlen() lunghezza di una stringa
  - strcmp() confronto fra due stringhe
  - strcpy() copia di una stringa in un'altra
    - fanno parte della libreria standard string.h
- Esistono molte altre funzioni di manipolazione di stringhe in ambiente C

## strlen()

- L'intestazione di questa funzione è:  
int strlen(char \*stringa);
- La funzione restituisce il numero di caratteri che compongono la stringa argomento, escluso il carattere terminatore
- Esempio:

**La lunghezza è 16**

```
main() {
 char *stringa = "Esempio di frase";
 printf("La lunghezza è %d", strlen(stringa));
}
```



## strcmp( )

- La funzione strcmp ha una intestazione del tipo:  
int strcmp(char \*str1, char \*str2);
- Le due stringhe passate come argomento vengono confrontate carattere per carattere
  - se sono uguali, il valore restituito è 0, altrimenti la funzione restituisce la differenza numerica tra i primi due caratteri responsabili della differenza
- Esempio:

```
main() {
 printf(" %d\t ", strcmp("salve","salve"));
 printf(" %d ", strcmp("valve","salve"));
}
```

0 3

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
85




## strcpy( )

- L'intestazione di strcpy è la seguente:  
char\* strcpy(char \*str1, char \*str2);
- La stringa puntata dal secondo parametro viene ricopiata, carattere per carattere, nello spazio di memoria della prima stringa, restituendo il puntatore alla nuova stringa
- Esempio:

```
main() {
 char stringa[20];
 strcpy(stringa, "tic tac, drin drin");
 printf("%s ", stringa);
 printf("%s", strcpy(stringa,"a b c"));
}
```

tic tac, drin drin a b c


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
86



## Esempio

```
/* Ritorna la lunghezza della stringa */
int strlen(char s[]) {
 int i = 0;
 while(s[i++] != '\0'); /* scandisce la stringa */
 return (i-1);
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
87



## Esempio

```
/* Legge una stringa in input di massimo size caratteri e
ritorna la lunghezza */
int getline(char s[], int size) {
 int c, i = 0;
 while((i < size-1) && ((c = getchar()) != EOF) && (c != '\n'))
 s[i++] = c;
 s[i] = '\0';
 return i;
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
88

## Array di puntatori

- Un array di puntatori è un array i cui elementi sono dei puntatori a variabili:
  - `int *arr_int[10]`
  - `arr_int[0]` contiene l'indirizzo della locazione di memoria contenente un valore intero
- I puntatori a caratteri vengono usati per rappresentare il tipo stringa, che non risulta definito nel linguaggio, e gli array di puntatori per rappresentare array di stringhe
  - `char *arr_str[10]`

## Array di puntatori a carattere

- Ad esempio:
  - `char *term[100];`
- indica che gli elementi di `term` sono dei puntatori a carattere, cioè `term[0]` è l'indirizzo di un carattere
- Ad esempio, `*term[0] = 'c'` indica che il contenuto della locazione di memoria puntata da `term[0]` è 'c';
- `term[7] = "Ciao"` invece indica che il contenuto di `term[7]` è il puntatore alla stringa "Ciao"

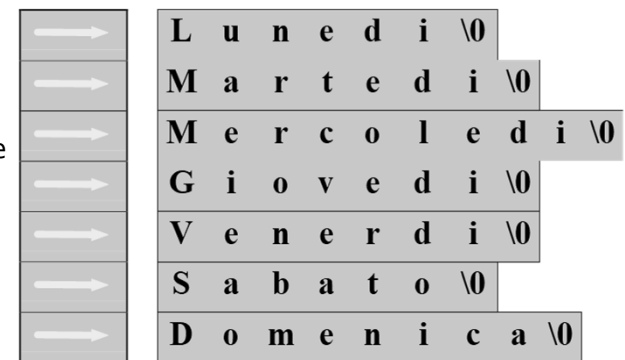
## Esempio

```
#include <stdio.h>
main() {
 const int GIORNI=7;
 char *giorni[GIORNI] = {"Lunedì", "Martedì", "Mercoledì",
 "Giovedì", "Venerdì", "Sabato", "Domenica"};
 int i;
 for(i=0; i<GIORNI; i++) printf("%s\n", giorni[i]);
}
```

```
Lunedì
Martedì
.....
```

## Esempio grafico

Rappresentazione  
a vettore di  
puntatori



## Argomenti sulla linea di comando

- Il main è una funzione a cui è possibile passare degli argomenti mediante i parametri in linea di comando
- Esiste una modalità ben precisa di passaggio dei parametri. In particolare, la funzione main riceve due argomenti:
  - `int main (int argc, char *argv[ ])`
- `argc` contiene il numero degli argomenti passati in linea di comando, mentre `argv` è un vettore di stringhe che contiene gli argomenti passati, uno per stringa

## Argomenti sulla linea di comando

- Gli argomenti passati in linea di comando devono essere separati da spazi bianchi
- Il nome del programma rappresenta il primo parametro passato, per cui l'argomento `argc` vale sempre almeno 1 e l'elemento `argv[0]` contiene sempre il nome del programma invocato
- Se `argc` vale 1, il main non ha argomenti, altrimenti `argv[1]` contiene il primo argomento ed `argv[argc-1]` contiene l'ultimo argomento
- Si assume che il contenuto di `argv[argc]` sia un puntatore nullo

## Esempio

- Il programma echo stampa a video i parametri passati
- ```
int main(int argc, char *argv[ ] ) {
    int i;
    for(i = 1; i < argc; ++i)
        printf("%s%s", argv[i], (i < argc-1) ? " " : "");
    printf("\n");
    return 0;
}
```

La ricorsione

- Il concetto di ricorsione nasce dalla possibilità di eseguire un compito applicando lo stesso algoritmo ad un dominio ridotto rispetto a quello originale e fondendo i risultati parziali
- Le funzioni C possono essere usate ricorsivamente, cioè una funzione può invocare se stessa sia direttamente che indirettamente
- Nella ricorsione è importante la condizione di uscita
 - il problema deve poter essere suddiviso in sottoproblemi più piccoli fino ad arrivare ad un sottoproblema banale di cui si conosce immediatamente la soluzione



Esempio di ricorsione

- Calcolo del fattoriale di un numero:

$$n! = n * (n-1) * (n-2) * \dots * (n - (n-1))$$

```
int fact(int numero) {
    if( numero == 1 )
        return 1;
    else
        return (numero * fact(numero - 1));
}
```



Esempio di ricorsione

- Calcolo del numero di Fibonacci:

$$f(n) = f(n-1) + f(n-2); \quad f(0) = 0; \quad f(1) = 1;$$

```
int fib(int numero) {
    switch(numero) {
        case 0: return 0;
        case 1: return 1;
        default: return (fib(numero-1) + fib(numero-2));
    }
}
```