LABORATORIO DI PROGRAMMAZIONE 2
Corso di laurea in matematica

Introduzione al linguaggio C

Marco Lapegna
Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

wpage.unina.it/lapegna

Evoluzione del C

il C e' un linguaggio di programmazione ad alto livello sviluppato agli inizi degli anni '70 del XX sec. presso i BELL Laboratories da Dennis Ritchie

E' estremamente potente (i sistemi operativi Unix e Linux sono scritti in C) e versatile (e' possibile sviluppare applicazioni in molti campi:

- industriale,
- scientifico,
- telecomunicazioni

Dennis Ritchie (sn) e Ken Thompson (dx) nel 1972

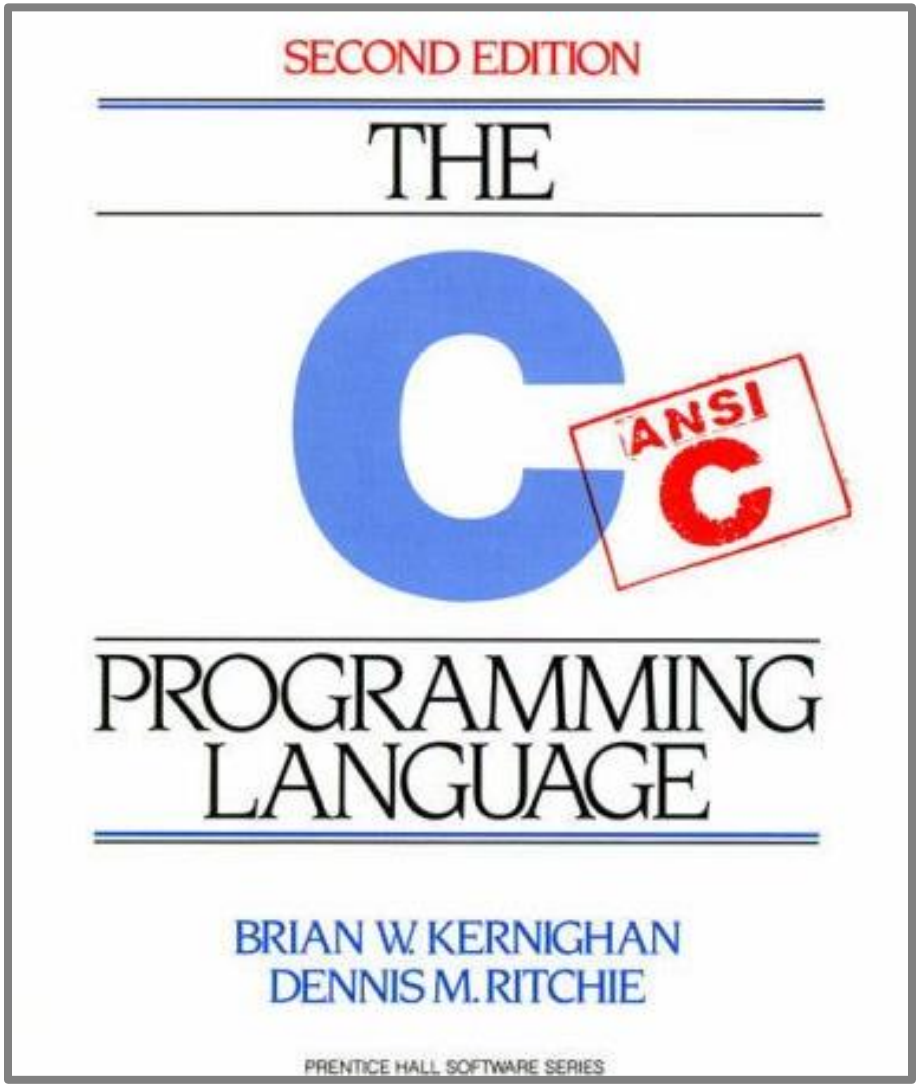
Evoluzione del C

L' ANSI C (1990) e' la principale versione standard del C, e garantisce una ampia portabilita' dei codici

Dal C sono derivati numerosi linguaggi di nuova generazione come

- C++,
- Java
- PHP

E' un linguaggio compilato, e quindi adatto anche al calcolo scientifico



Il principale manuale del linguaggio C

Passi per risolvere un problema con il calcolatore

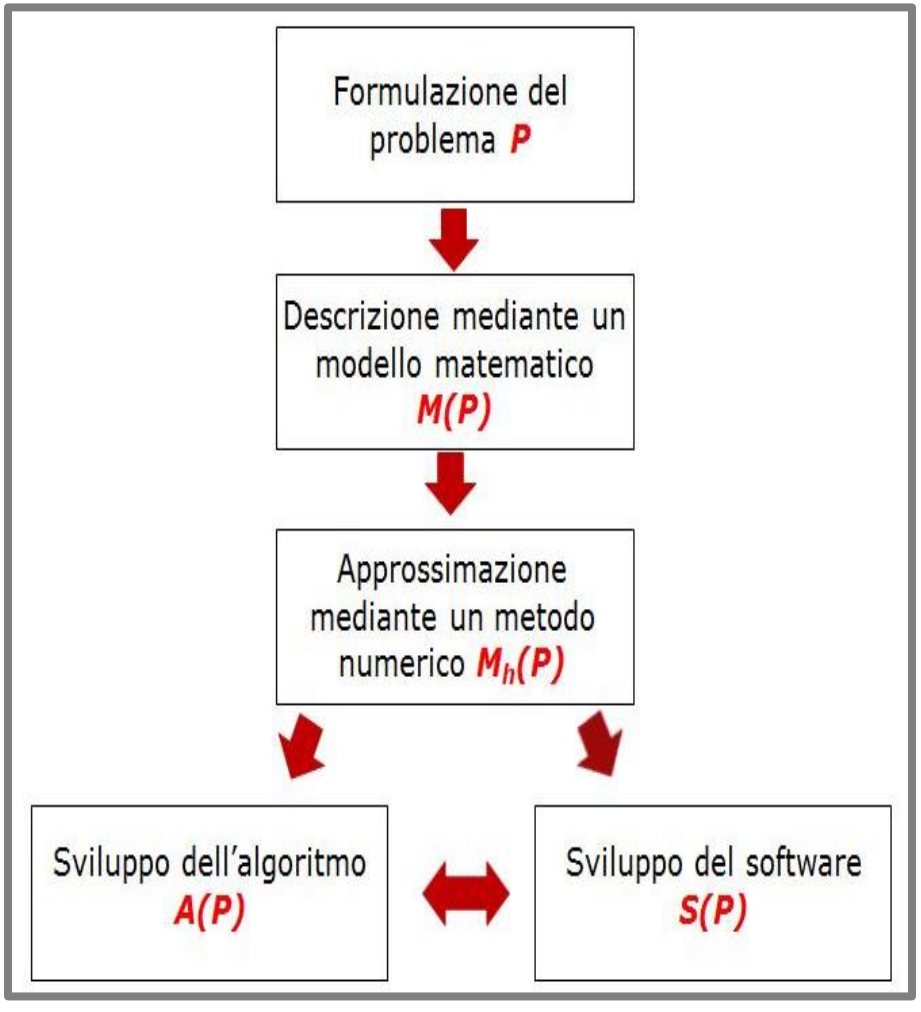
Un programma deve essere visto come la

Traduzione di un algoritmo in un linguaggio di programmazione

Di nostra competenza il passaggio da

A(P) algoritmo
al
S(P) software

Importante solo rispettare la sintassi



passi per la risoluzione di un problema scientifico con il calcolatore

Inizio e fine programma

- Ogni programma C e' compreso tra le parentesi graffe { } che seguono l'istruzione main
- main e' sempre il nome del programma principale (obbligatorio)

Pascal Like

```
begin nomealgoritmo

.....
.....
.....
.....

end nomealgoritmo
```

Inizio e fine in Pascal-like

- Analogo di begin ed end per il Pascal Like

C

```
main ( ) {

...
...

}
```

Equivalente C

- In linguaggio C ogni istruzione termina con ;
- E' possibile inserire piu' istruzioni su una stella riga (ovviamente separate da ;)

Dichiarazione delle variabili

- 5 tipi di dati fondamentali
- Il C e' Case Sensitive (fa differenza tra maiuscole e minuscole)
- Obbligatorio dichiarare le variabili
- Altri tipi meno usati: unsigned, long ,…
- L'utilizzo del tipo logico (bool) richiede #include<stdbool.h>

Pascal Like

```
.....
var: M, N: integer
var: X, Y: real
var: Z, W: double
var: A, B: character
var: R, T: boolean
.....
```

Dichiarazione variabili in Pascal-like

C

```
int  M, N ;
float  X, Y ;
double Z, W;
char   A, B ;
bool  R, T;
```

Equivalente C

Lettura, definizione e stampa delle variabili

- scanf(…) definisce una o piu' variabili mediante l'unita' di input (es. la tastiera)
- printf(…) stampa il contenuto di una variabile sull'unita' di output (es. il monitor)
- scanf e printf sono funzioni intrinseche del C che vanno dichiarate. La dichiarazione e' presente nel file stdio.h che va "incluso" nel programma con l'istruzione #include<…>
- l'operatore % specifica il tipo di variabile da leggere/stampare:
  d : interi f : reali e d.p. c : carateri
- gli operatori & nell'istruzione scanf saranno chiari piu' avanti …..
- L'operatore = indica una operazione di assegnazione di un valore ad una variabile

Pascal Like

```
.....
Read A, B, C
X = A+B+C
Print X
.....
```

Lettura, scrittura e assegnazione in P-L

C

```
#include<stdio.h>
main() {
int A, B, C, X;
...
scanf (" %d %d %d ", &A, &B, &C);
X = A + B + C;
printf("somma = %d \n", X);
...
```

Equivalente C

commenti

I commenti sono linee di testo inserite nel codice che non vengono tradotte dal compilatore

Hanno lo scopo di migliorare la leggibilita' dei programmi, descrivendo le sezioni del programma o anche gruppi di istruzioni

E' buona norma inserire numerose linee di commento (anche fino al 50% delle linee dl programma)

In C le linee di commento iniziano con //

Pascal-like

```
Begin prova
.....
(*  Questo e' un commento *)
...

End prova
```

Commenti in P-L

C

```
main( ){
.....
//   Questo e' un commento
...

}
```

Equivalente C

Struttura di selezione

- Esatta corrispondenza con il Pascal Like
- le istruzioni che compongono i rami della struttura sono racchiusi tra parentesi graffe
- { … }
- Disponibile anche senza il ramo ELSE

Pascal Like

```
.....
If (condizione)  then
                .....
                .....
Else
                .....
                .....
Endif
.....
```

If-then-else in pascal-like

C

```
if ( condizione ) {

....

} else {

....

}
```

Equivalente in C

Operatori logici e relazionali

- Vengono eseguiti prima gli operatori relazionali

  > < == != >= <=

- Poi quelli logici

  ! && ||

- ! (not)
- && (and)
- || (or)

Pascal Like

```
Boolean L

.....

L = (A > B)  or  (C >= D)

L = (X == Y)  and  (Z /= W)

L = not L
```

Esempi di operatori logici e relazionali in P-L

C

```
#include<stbool.h>
main() {
bool L;
.....
L = (A > B) || (C >= D);
L = (X == Y) && (Z != W);
L = !L

}
```

Equivalente C

Struttura di iterazione for

- Tre campi sempre obbligatori
  - Inizializzazione dell'indice
  - Condizione di validita' dell'indice !!
  - Incremento dell'indice
- Istruzioni da ripetere racchiuse da parentesi graffe {… }
- Piu' strutture innestate richiedono indici distinti
- nel caso di incremento di 1 si puo' usare l'operatore i++ , equivalente a i=i+1
  (es. for (i=1; i < N; i++))

Pascal Like

```
for i = 1 to N step K

      ......
      ......

endfor
```

For-endfor in Pascal-like

C

```
for ( i =0; i < N; i=i+k) {

......
......

}
```

Equivalente C

Struttura di iterazione repeat

- Ripete piu' volte le istruzioni tra repeat e until
- Quanto la condizione risulta **vera** la struttura di iterazione **termina**
- Non esiste una traduzione "diretta"
- Realizzato mediante do-while
- poiche' il while continua l'esecuzione quando la condizione e' vera, in questo caso e' **necessario negare la condizione di uscita**

Pascal Like

```
repeat
        .....
        .....
until (condizione)
```

Repeat-until in Pascal-like

C

```
do {
.....
.....
} while ( ! (condizione) ) ;
```

Equivalente C

Struttura di iterazione while

- Ripete piu' volte le istruzioni tra while e endwhile
- Quanto la condizione risulta **vera** la struttura di iterazione **continua**
- traduzione diretta del pascal-like

Pascal Like

```
while (condizione)
        .....
        .....
endwhile
```

While-endwhile in pascal-like

C

```
.....
while (condizione){
...
...
}
```

Equivalente C

Gli array

- in fase di dichiarazione e' necessario specificare la dimensione
- successivamente non e' possibile modificare la dimensione
- e' possibile fare riferimento a ciascuna componente tramite un indice
- La prima componente ha indice 0 (zero)
- in memoria le componenti di un array occupano locazioni di memoria consecutive
- gli array 2-dimensionali sono memorizzati per righe in locazioni consecutive

Pascal Like

```
var: a(100): array of real
var: X(10,10): array of integer

...
...

for i=1 to 10
    read a(i)
endfor
```

Dichiarazione e lettura di un array in P-L

C

```
float A[100];
int  X[10][10];

...
...
for (i=0; i<10; i++){
   scanf("%f", &A[i]);
}
```

Equivalente C

Sottoprogrammi FUNCTION

- Unico tipo di procedura disponibile in C
- Invocata attraverso il nome, puo' ritornare direttamente un valore al programma chiamante attraverso il nome
- la function deve essere dichiarata con un tipo, che coincide con il tipo ritornato dalla function (se non ritorna niente il tipo e' void).
- Nella dichiarazione della funzione si dichiara anche il tipo dei dati di input
- Gli argomenti della function sono tutti argomenti di input
- L'esecuzione del programma chiamante riprende dopo il termine della subroutine

Pascal Like

```
begin mainprogram
var: A, B, C: integer
read A, B

somma (in: A, B; out: C)

print C
end
```

Chiamata di una procedura in Pascal-like

C

```
#include<stdio.h>
main(){
int A, B, C;
int somma(int, int);
scanf("%d %d", &A, &B);
C=somma(A,B);
printf(" %d \n", C);
}
```

Equivalente C

Sottoprogrammi FUNCTION

- il passaggio delle informazioni e' **per valore** (della variabile)
- la function opera su una copia delle locazioni di memoria delle variabili del programma chiamante
- L'istruzione return, ritorna il valore di output al programma chiamante (opzionale se la funzione e' void)
- E' possibile utilizzare nomi diversi nelle function e nel programma chiamante
- Nella testata vanno dichiarati anche i dati di input
- Osservazione. Le function ritornano sempre un solo valore (oppure nessuno se la funzione e' void)

Pascal Like

```
procedure somma(in: X, Y; out: Z)
var: X, Y, Z: integer

Z = X + Y

end somma
```

Procedura in Pascal-like

C

```
int somma( int X, int Y){
int Z;
Z = X+Y;
return Z;
}
```

Equivalente C

Esempio: scambio di 2 variabili

Questo programma non funziona !!

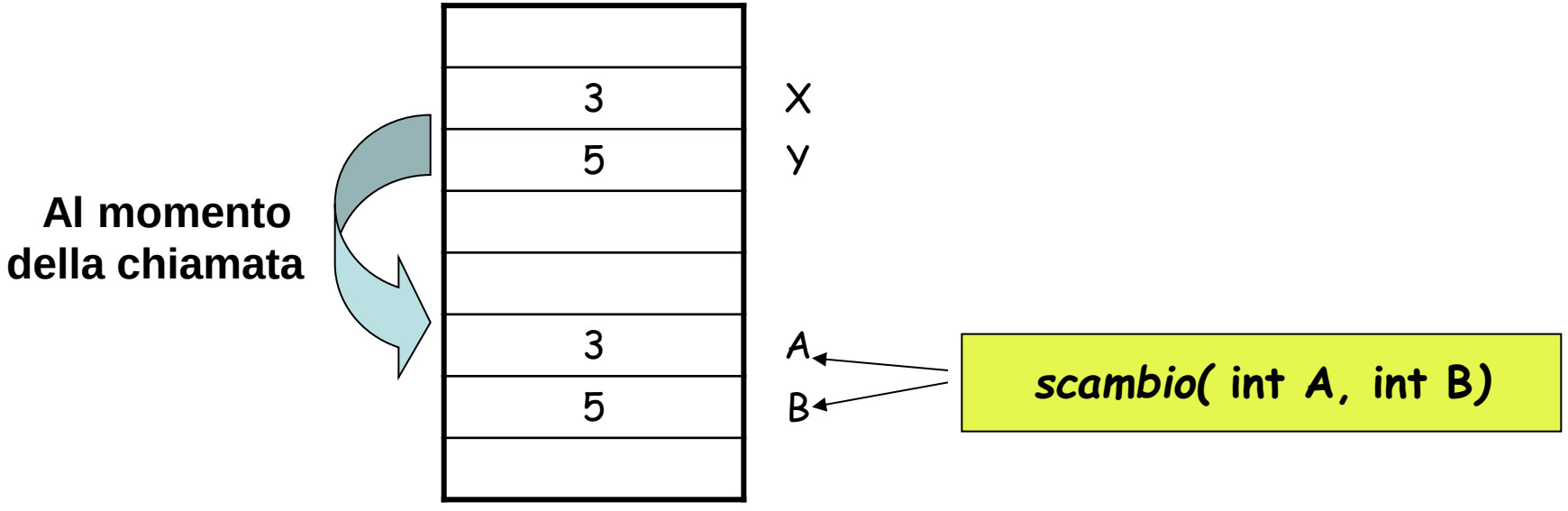
perche?

```
#include <stdio.h>
/*  Prototipo della funzione  */
main( ) {
    void scambio( int,  int);
     int  X, Y;
     X=3; Y=5;
     scambio(X, Y);
     printf("X= %f  Y= %f \ n", X, Y);
}


void scambio (int A, int B) {
   int  tmp;
   tmp=A;   A=B;   B=tmp;
}
```

Cosa avviene in memoria?

In C, gli argomenti sono passati per valore, quindi i parametri formali risiedono in locazioni di memoria differenti da quelle utilizzate dai parametri attuali.



Scambiare A e B NON equivale a scambiare x e y !!

Function con piu' valori di ritorno

- Come fare se si vuole una function che ritorna 2 o piu' valori?
- IDEA: utilizzare lo stesso meccanismo del Fortran e passare l'indirizzo delle variabili. In tal modo le function operano direttamente sulle stesse locazioni di memoria
- Nel main, per passare l'indirizzo delle variabili X e Y si usa l'operatore &
- Nella function, A e B contengono l'indirizzo delle variabili X e Y del main. Per accedere a X e Y si usa l'operatore *
- Esempio: function per lo scambio di due variabili A e B
  - Input: indirizzi delle variabili X e Y
  - Output: niente (funzione void)
- Ora dovrebbe essere chiara la presenza di & nelle istruzioni scanf..

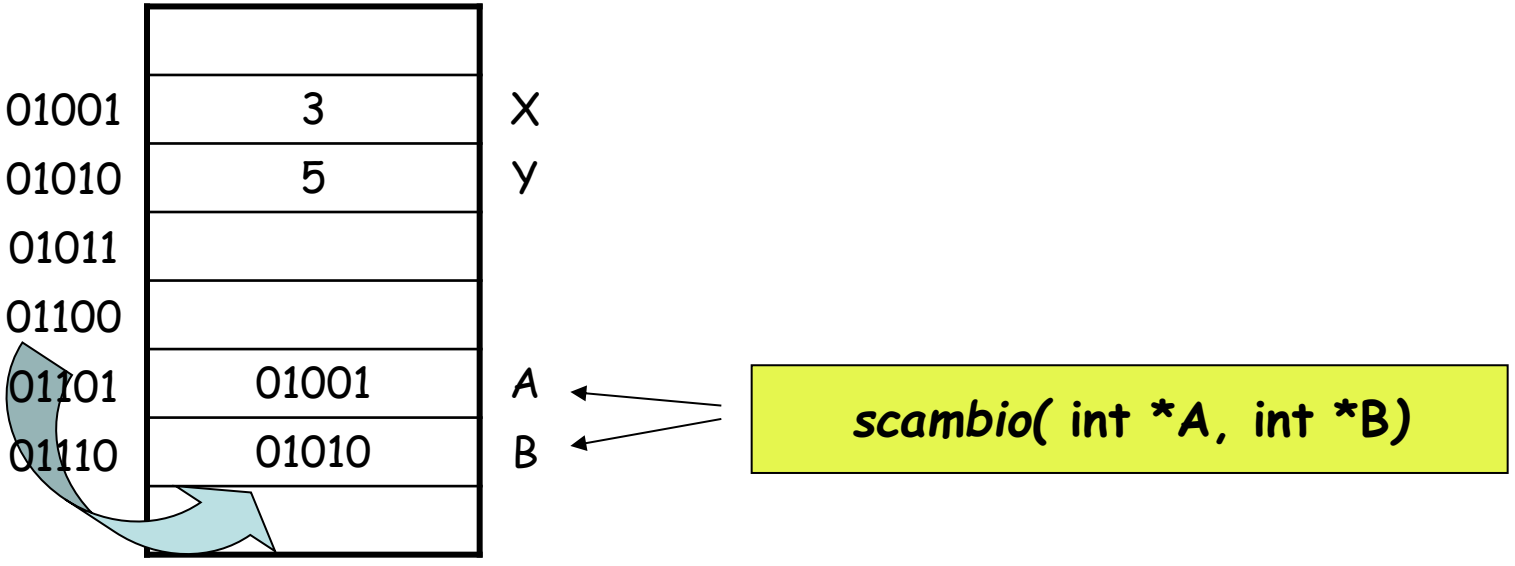
```
#include <stdio.h>
/*  Prototipo della funzione  */
main( ) {
   void scambio( int *,  int *);
    int  X, Y;
    X=3; Y=5;
    scambio(&X, &Y);
    printf("X= %f  Y= %f \ n", X, Y);
}
```

Programma chiamante

```
void scambio (int *A, int *B) {
  int  tmp;
  tmp = *A;   *A = *B;   *B = tmp;
}
```

Function per lo scambio

Cosa avviene in memoria?



Al momento della chiamata

Vengono scambiate le variabili puntate da A e B (cioe' X e Y)!!

riassumendo

- L'indirizzo (o puntatore) di una variabile si ottiene anteponendo il carattere & al nome della variabile
  - Es: &a e' l'indirizzo della variabile a

- Una variabile puntatore si dichiara anteponendo il carattere * al nome della variabile puntatore
  - Es: int *tp dichiara la variabile tp come variabile puntatore ad una variabile intera

- Il valore di una variabile puntata da una variabile puntatore si ottiene con un *
  - Es: *tp e' il valore della variabile puntata da tp

- Assegnazioni
  - Es: tp = &a assegna a tp l'indirizzo di a
  - Es: b = *tp assegna a b il valore della variabile puntata da tp

Altri esempi

• l'istruzione

```
int  a, b,  *tp ;
```

dichiara che a, b sono variabili intere e che tp e' una variabile puntatore ad una variabile intera

• l'istruzione

```
a = 3;
```

assegna il valore 3 alla variabile a

• l'istruzione

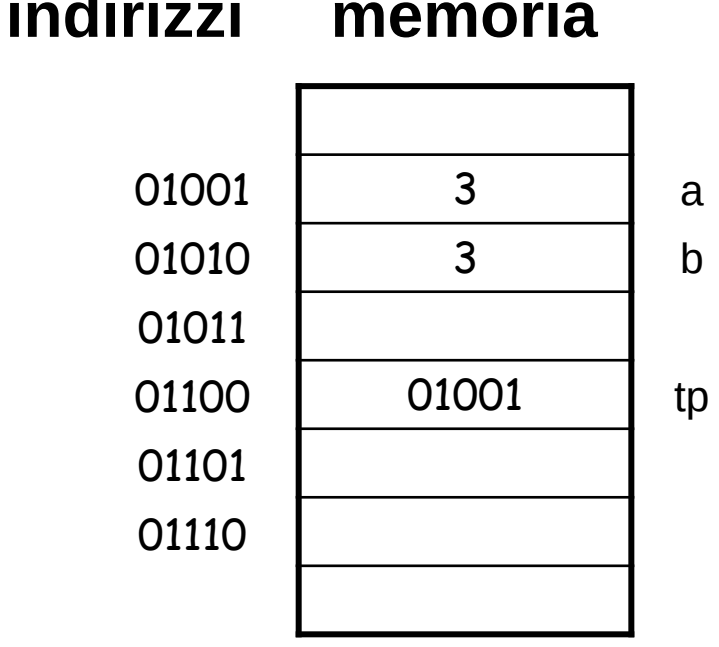
```
tp  =&a
```

Assegna l'indirizzo 01001 a tp

• l'istruzione

```
b = *tp
```

assegna a b il valore della variable puntata da tp (cioe' a)



Passaggio di un array ad una function

Il nome di un array e' l'indirizzo del primo elemento

Quindi

attraverso il nome, viene passato l'indirizzo della prima componente

(analogo del Fortran)

- Allocazione di array in locazioni consecutive
- Passaggio per indirizzo (la function opera sulle stesse locazioni di memoria)

Da queste informazioni la function e' in grado di "ricostruire" tutto l'array

Nella function e' possibile evitare di dichiarare gli array specificando una dimensione fissata

C

```
int  A[9];
void funz( int, int [] );

n = 3;
...
funz( n, A);
...
```

Passaggio di un array ad un function in C

C

```
void funz (int  n,  int a[]){
int   i;
for (i=0; i<n; i++){
   printf(" %d" ,a[i]);
}

}
```

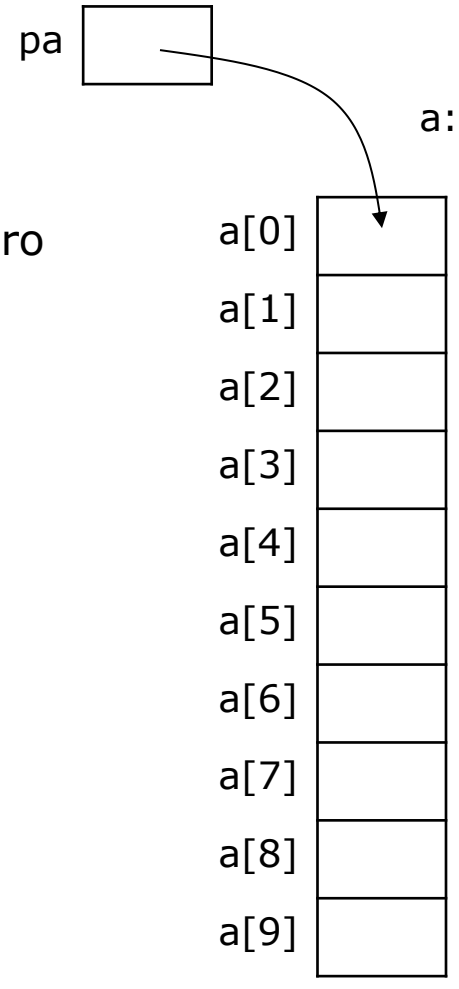
Function C che stampa un array

Per capire meglio...

L'istruzione `int a[10] , *pa;`

Dichiara

- Un array `a` di 10 elementi
- Una variabile `pa` che puo' contenere l'indirizzo di un intero



Le istruzioni

`pa = a` e `pa = &a[0]`

SONO EQUIVALENTI !!

Inoltre, poiche' `pa` e' una variabile hanno senso

- `pa+i` (e' l'indirizzo di `a[i]`)
- `*(pa+i)` (e' il valore di `a[i]`)

Infine, poiche' il nome `a` e' l'indirizzo di `a[0]`

- `a+i` e' l'indirizzo di `a[i]`
- `*(a+i)` e' il valore di `a[i]`

Array 2-d e function

- Gli array 2-d sono memorizzati per righe
- E' necessario specificare la leading dimension (numero massimo di colonne)
- Nella dichiarazione della funzione nel programma chiamante e' possibile indicare con (*)[] la presenza di un puntatore ad un array 2-d
- Nella testata della function si usa la leading dimension per specificare il numero di colonne dell'array
- La leading dimension deve precedere il nome dell'array nella lista degli argomenti

C

```
int  A[9][9];
void funz( int, int (*)[ ] , int);

ld =9;
n = 3;
...
funz( 9, A, n);
...
```

Esempio di chiamata di funzione con array 2-d

C

```
void funz(int  ld, int a[][ld], int n){
  int i, j;
  for (i=0; i<n, i++){
  for (j=0; j<n, j++){
     printf("%d ", a[i][j]);
  } printf("\n");  }
}
```

Funzione C che stampa un array 2-d

Esempio: trasposta di matrice (function)

```
void trasp( int  ld,  int   a[][ld],  int n) {

   int  i,  j, t;

   for (i=1;  i<n;  i++){
     for (j=0;  j<i;  j++){
         t = a[i][j];
         a[i][j] = a[j][i];
         a[j][i]  = t;
    }}

}
```

Dichiarazione della matrice con la leading dimension

Utilizzare tale function per trasporre il minore di ordine n=4 a partire da a[3][3] in una matrice di ordine 9.

Esempio: trasposta di matrice (main)

```
#include <stdio.h>
main( ) {
    int  i,  j, a[9][9], n;
/*  Prototipo della funzione  */
   void trasp( int,  int (*)[],  int );

    for (i=0; i<9; i++){
    for (j=0; j<9; j++){
        if (i<=j) {      a[i][j] = 10*i+j;
        }  else {        a[i][j] = 0;
        }
    }}

    n=4;
    trasp( 9, (int (*)[])&a[3][3],  n);

    for (i=0; i<9;  i++){
    for (j=0; j<9; j++){
        printf( " %3d ", a[i][j] );
     }printf("\n");  }

}
```

Inizializza matrice quadrata triangolare superiore con di ordine 9

Traspone il minore di ordine n=4 a partire da a[3][3]. Il primo argomento e' la LD

(int (*)[]) &a[3][3]
e' la conversione dell'indirizzo di a[3][3] in un generico
indirizzo di un array 2-d di interi

Stringhe di caratteri

Ogni variabile di tipo carattere puo' contenere un solo carattere

Per definire stringhe di caratteri (sequenze di 2 o piu' caratteri) e' necessario dichiarare array di caratteri

e' possibile leggere e stampare stringhe di caratteri con le usuali funzioni scanf e printf utilizzando l'operatore %s

Poiche' le stringhe sono array di caratteri non e' necessario l'operatore & nella funzione scanf

```
#include<stdio.h>
main( ) {
   char nome[10];
  …
   scanf("%s", nome);
   printf("%s \n", nome);
  …
}
```

Lettura e stampa di stringhe di caratteri

Funzioni intrinsiche

Il C dispone di una libreria di function intrinsiche per le piu' comuni funzioni matematiche e/o di utilita

- Y = sqrt(X)
- Y = abs(X)
- Y = tan(X)

- Y = sin(X)
- Y = cos(x)

- Y = log(X)
- Y = log10(X)
- Y = exp(X)

- E tante altre…

- Necessario includere <math.h>
- Compilare con opzione -lm

```
#include<stdio.h>
#include<math.h>
main () {
float x, y, pi, h;

pi = acos(-1);
h = pi/10.;

for (i=1; i<=10; i++){
   x = 0 + i*h;
   y = sin(x);
   printf(" %f %f \n", x, y);
}

}
```

Semplice programma per il calcolo di 11 valori di sin(x) nell'intervallo [0,3.14]

Passaggio di una function ad una function

Problema:

Siano due funzioni

fun1 = x+1 e fun2 = 1-x

Si vuole costruire una function funquadro
che calcola il quadrato
delle funzioni passate come argomento

Chiamata a funquadro

y = funquadro(x, fun1)

Function passata come argomento!

C

```
#include<stdio.h>
main() {
double  x, y,  fun1(double),   fun2(double);
double funquadro ( double, double(double) );

x = 5;

y = funquadro(x, fun1);
printf( "%f  \n",  y);

y = funquadro(x, fun2);
printf( "%f  \n",  y);

}
```

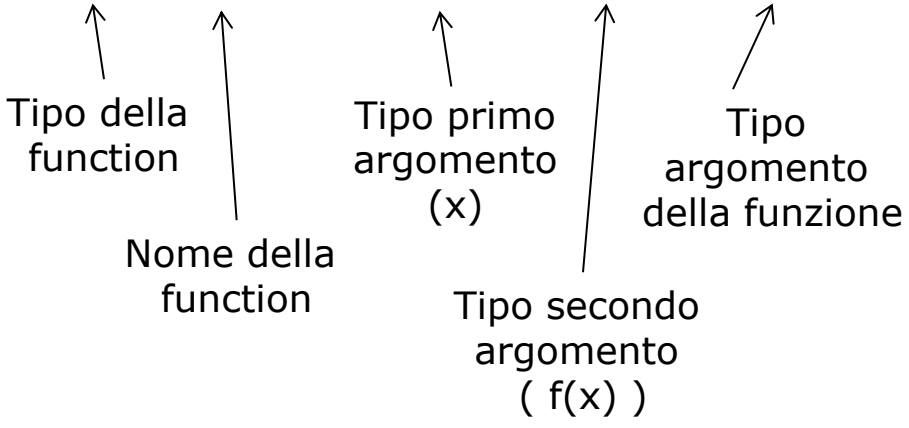
Esempio di programma chiamante che utilizza funquadro per calcolare il quadrato di fun1 e fun2

Analogamente agli array,
i nomi delle function sono dei puntatori

Passaggio di una funzione ad una function

Dichiarazione di funquadro nel main

double funquadro (double, double(double))




```
double funquadro (double x, double f(double) ){
double z;
z = f(x)*f(x);
return z
}

double fun1(double x){
double y;
y = x+1;
return y
}

double fun2(double x){
double y;
y = 1-x;
return y
}
```

Funzioni fun1, fun2 e funquadro

Testata di funquadro

double funfun (double x , double f (double))

(con la specifica del nome
degli argomenti x e f(x))

argomenti del main

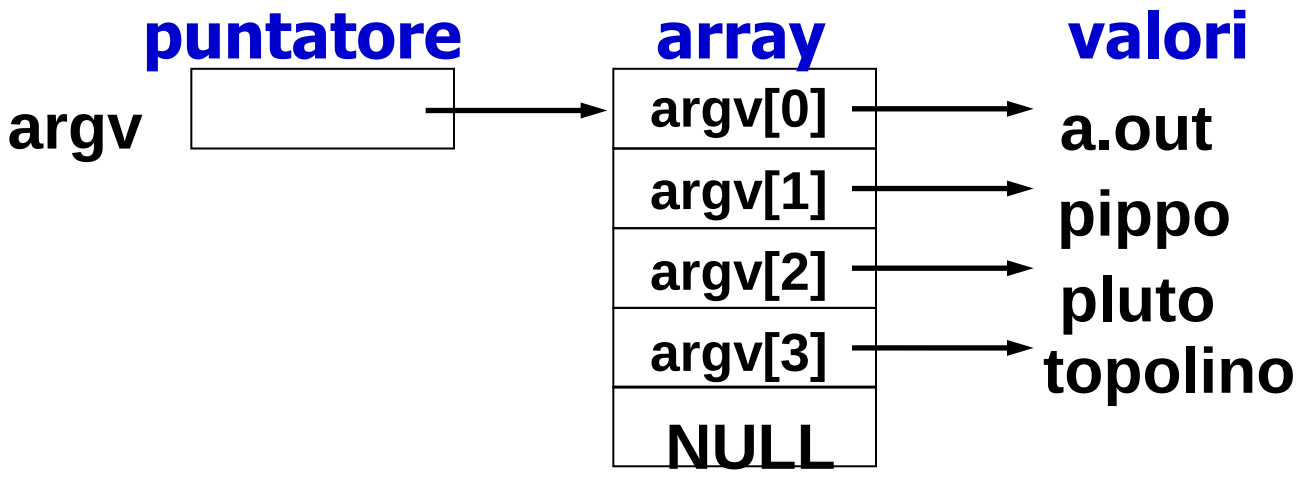
```
#include <stdio.h>

int   main( int argc, char *argv[ ] )
{
   int   i;

   for (i = 0; i < argc; i++)
                  printf("argv[%d]: %s\n", i, argv[i] );

   exit(0);
}
```

ergc e argv



argc e' un intero (numero di argomenti sulla linea di comando, compreso il comando stesso)

argv e' un array di puntatori a stringhe di carattere (gli argomenti sulla linea di comando)

esempio (cont.)

```
[lapegna%] cc program.c
[lapegna%] a.out  pippo  pluto  topolino
argv[0]: a.out
argv[1]: pippo
argv[2]: pluto
argv[3]: topolino
[lapegna%]
```

funzione atoi() ASCI to integer

utile per convertire stringhe di caratteri in interi da utilizzare all'interno del main

```
#include <stdio.h>

int   main( int argc, char *argv[] )
{
  int   RES ;

  RES =  atoi(argv[1])  +  atoi(argv[2])  +  atoi(argv[3]);
  printf(" risultato = %d \n", RES);

}
```

```
[lapegna%] cc program.c
[lapegna%] a.out  5  12  8
risultato = 25
[lapegna%]
```

esempio (cont.)

```
[lapegna%] cc program.c
[lapegna%] a.out  5  12  8
risultato = 25
[lapegna%]
```

Strutture dati 'avanzate'

Principali caratteristiche degli array:

PRO:
- Efficienza (accesso diretto tramite indice)

CONTRO
- Dimensione prefissata non modificabile
- Elementi omogenei (tutti dello stesso tipo)

OBIETTIVO

Definire strutture con elementi
di tipo disomogeneo

ES: uno studente
- Nome (char)
- Cognome (char)
- Voto matematica (int)
- Voto informatica (int)
- Media (float)

studente

nome	Gennaro
cognome	Esposito
matematica	25
informatica	28
media	26,5

Una struttura dati che contiene informazioni di tipo disomogeneo

Le strutture (struct) in C

Il C possiede l'istruzione

struct

per definire nuovi tipi di variabili strutturate

E' necessario specificare:

- Il nome del nuovo tipo di variabile strutturata
- Il tipo ed il nome dei campi

```
main ( ){

// specifica del nuovo tipo di variabile strutturata (studente)
 struct studente{

// specifica dei campi della struttura
    char nome[10];
    char cognome[10];
    int matematica;
    int informatica;
    float media;
};

// dichiarazione di una variabile strutturata di tipo studente
 struct studente  stud1;
...
...
```

Cio' descrive solamente l'aspetto delle variabili strutturate !!

Specifica di una nuova variabile strutturata di tipo "studente" e dichiarazione di una variabile (stud1) di tipo studente

Successivamente e' necessario dichiarare

- I nomi delle variabili strutturate del nuovo tipo

Le strutture (struct) in C

Una volta dichiarata una variabile strutturata, come accedere ai singoli campi?

I campi della struttura vengono individuati con:

nomevariabile.nomecampo

Esempio:

- stud1.nome
- stud1.cognome
- stud1.matematica
- stud1.informatica
- stud1.media

```
...
// lettura
scanf("%s", stud1.nome);
scanf("%s", stud1.cognome);
scanf("%d", &stud1.matematica);
scanf("%d", &stud1.informatica);
scanf("%f",  &stud1.media);

// stampa
printf("%s", stud1.nome);
printf("%s", stud1.cognome);
printf("%d", stud1.matematica);
printf("%d", stud1.informatica);
printf("%f",  stud1.media);
```

Frazione di codice C che legge e stampa i campi di una variabile strutturata

Passaggio di una struct ad una function

Problema:

Sia una struct

stud1

Si vuole costruire una function stampastruct che stampa il contenuto della struct passata come argomento

Per passare una struttura ad una funzione e' necessario passare un puntatore alla struttura

(in maniera analoga ai puntatori di variabili ordinarie)

```
// dichiarazione di stampastruct
 void stampastruct(struct studente *);

// dichiarazione di stud1 e di un puntatore a struct
 struct studente stud1,  *sp;

// lettura di stud1
 scanf("%s", stud1.nome);
 …

// definizione del puntatore
 sp = &stud1;

// passaggio a  stampastruct
   stampastruct(sp);

…
```

Esempio di programma chiamante che utilizza stampastruct per stampare una struttura stud1

Passaggio di una struct ad una function

Dichiarazione di stampastruct nel main

```
void stampastruct ( struct studente *)
```

Tipo della function

Nome della function

Tipo argomento (puntatore a struct studente)

```
void stampastruct(struct studente *sp){

    printf("%s ", (*sp).nome);
    printf("%s ", (*sp).cognome);
    printf("%d ", (*sp).matematica);
    printf("%d ", (*sp).cognome);
    printf("%f ", (*sp).media);

}
```

Funzione stampastruct

Testata di stampastruct

```
void  stampastruct (struct studente *sp )
```

(con la specifica del nome dell'argomento sp)

Problema:

Il nuovo tipo di dato strutturato e' stato definito nel main e cio' non e' visibile nella funzione stampastruct

Per ovviare a tale problema e' sufficiente dichiarare il nuovo tipo di struttura prima

- Della funzione main
- Della funzione stampastruct

In questo modo la nuova struttura dati e' visibile da tutte le funzioni che seguono

```
#include<stdio.h>
struct studente{
char nome[10];
char cognome[10];
int matematica;
int informatica;
float media;
}

main ( ){
 …
 …
}

void stampastruct (struct studente *sp ) {
 …
 …
}
```

Visibilita' del nuovo tipo di variabile strutturata struct studente

In alternativa

E' possibile inserire dichiarazioni "globali" che devono essere utilizzate da piu' funzioni (anche presenti in file differenti) in appositi file chiamati

Header file

Si riconoscono per l'estensione .h e si utilizzando con l'istruzione

#include "header file"

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

main ( ){
 …
 …
}

void stampastruct (struct studente *sp ) {
 …
 …
}
```

File contenente programma chiamante e function

```
struct studente{
char nome[10];
char cognome[10];
int matematica;
int informatica;
float media;
};
```

File contenente struttura.h contenente la dichiarazione della struttura

osservazione

Notazione alternativa per i puntatori a struttura

Sia sp puntatore a struttura :

struct studente *sp

allora

(*sp).nome

e' equivalente

sp->nome

```
void stampastruct(struct studente *sp){
   printf("%s ", sp->nome);
   printf("%s ", sp->cognome);
   printf("%d ", sp->matematica);
   printf("%d ", sp->cognome);
   printf("%f ", sp->media);
}
```

Versione alternativa di stampastruct

Array di strutture

Una volta dichiarata un nuovo tipo di variabile strutturata e' ovviamente possibile dichiarare array di tale tipo

Esempio:

Classe di 10 studenti

Per ognuno dei quali si vuole

- Nome
- Cognome
- Voto matematica
- Voto informatica
- Media

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

main () {
   // dichiarazione di un elenco di 10 studenti
   struct studente elenco[10];
…
   // stampa dei cognomi
   for (i=0; i<10; i++){
      printf("%s", elenco[i].cognome);
   }
…
}
```

Stampa dei cognomi di un elenco di 10 studenti

FINE

Questo lucido, ed i successivi,
non fanno parte della lezione

Inserire titolo slide

Usare questa slide per inserire del testo, un'immagine e un video.

Digitare qui il testo.

Non superare i 400 caratteri spazi inclusi.

Inserire a lato l'immagine, cliccando sulla cornice.

Usare il campo note di PowerPoint per inserire link e risorse collegate alla presente slide o esplicitare esigenze di formattazione.

Per registrare un commento audio direttamente sulla slide:

Menu>Inserisci>Filmati e audio> Registra suono…

Nome: "Slide+numero"

Didascalia immagine Non superare i 90 caratteri spazi inclusi.

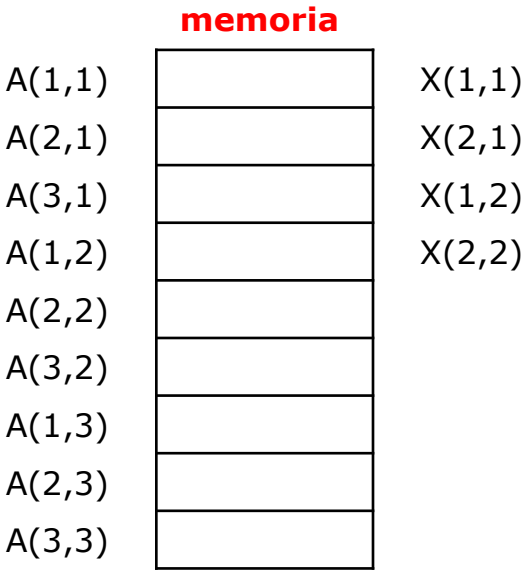
Didascalia filmato Non superare i 90 caratteri spazi inclusi.

```
PROGRAM MAIN
INTEGER :: N, A(3,3)

…
N=2
CALL SUBMAT(N,A)
..
```

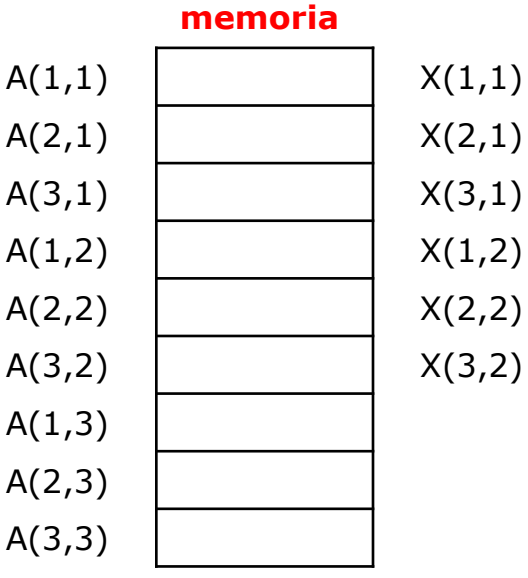
```
SUBROUTINE SUBMAT (M,X)
INTEGER :: M, X(M,M)

…
```



```
PROGRAM MAIN                          SUBROUTINE SUBMAT (LD,M,X)
INTEGER :: N, A(3,3)                  INTEGER :: M, X(LD,M)

...                                   ...
N=2
LD=3
CALL SUBMAT(LD,N,A)
..
```



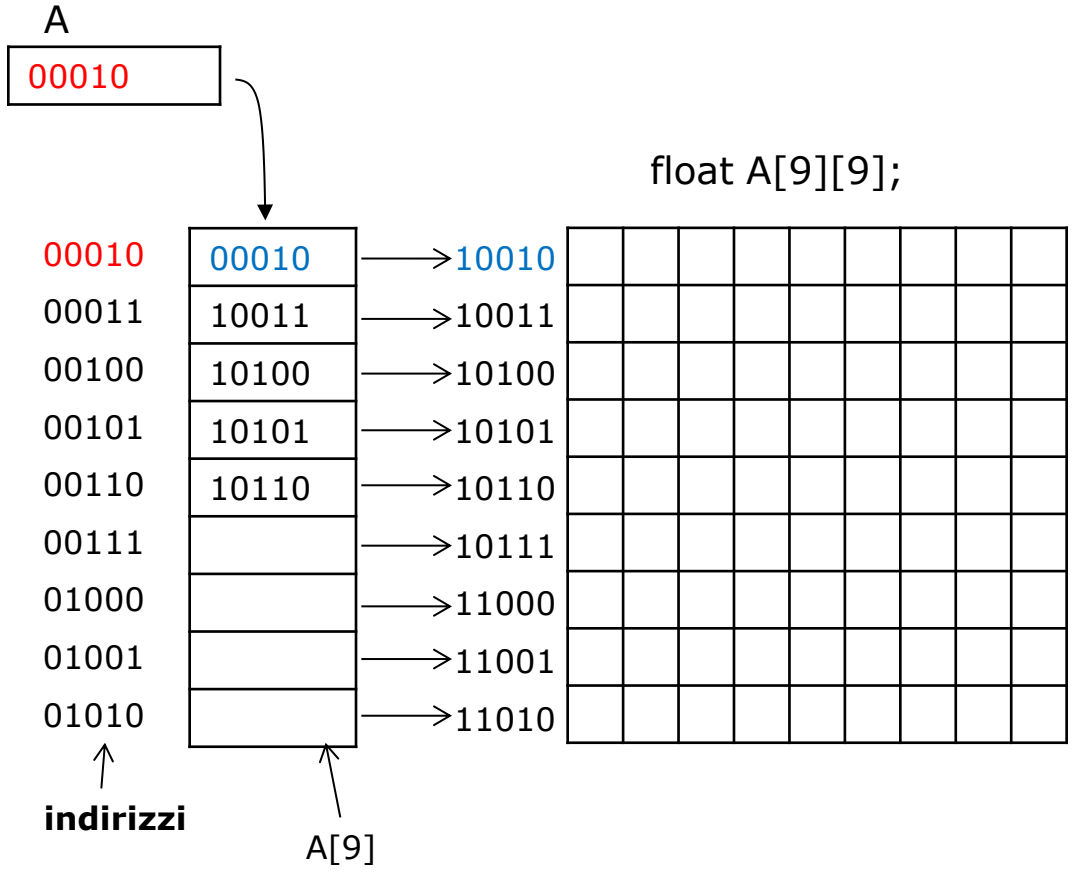
```
PROGRAM MAIN
REAL:: X, Y, PI, H

PI = ACOS(-1)
H = PI/10.

DO I = 0, 10
        X=0 + i*H
        Y = SIN(X)
        PRINT*, X, Y
ENDDO

END
```

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)
A(6)
A(7)
A(8)



C

```
main ( ) {
char nome[10];

scanf (“%s”, nome);

printf(“%s”, nome);

}
```

C

```
void funz (int  n,  int a[]){
int   i;
for (i=0; i<n; i++){
   printf(" %d" ,a[i]);
}

}
```

```
main (){

// dichiarazione di un elenco di 10 studenti
struct studente elenco[10],  *sp;
…

for (i = 0;  i<10;  i++){
   printf (“%s”,  elenco[i].cognome);
}
```

studente

nome	Gennaro
cognome	Esposito
matematica	25
informatica	28
media	26,5

```
#include<stdio.h>
#include<math.h>
main () {
float x, y, pi, h;

pi = acos(-1);
h = pi/10.;

for (i=1; i<=10; i++){
   x = 0 + i*h;
   y = sin(x);
   printf(" %f %f \n", x, y);
}

}
```