Linguaggio di programmazione

C

Linguaggio C

- UN PO’ DI STORIA
  - definito nel 1972 (AT&T Bell Labs) per sostituire il linguaggio Assembler
  - Usato per riscrivere il kernel di Unix dal gruppo di Ken Thompson, Dennis Ritchie nel 1973
  - prima definizione precisa: libro “The C Programming Language” (1978) di Kernigham & Ritchie
  - prima definizione ufficiale: ANSI (1983)

Linguaggio C

- Caratteristiche:
  - linguaggio sequenziale, imperativo, strutturato
  - usabile anche come linguaggio di sistema
  - adatto a software di base, sistemi operativi, compilatori, ecc.
  - portabile, efficiente, sintetico
  - ma a volte poco leggibile...

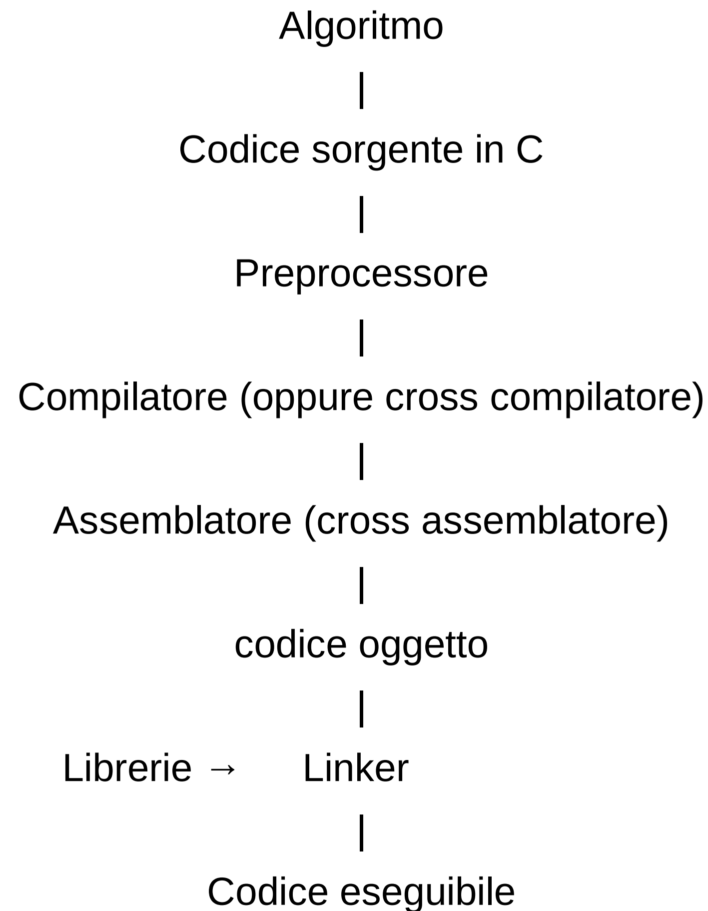
Linguaggio C

- Caratteristiche:
  - portabilita’
  - non e’ un linguaggio fortemente tipizzato
  - consente la gestione di bit, byte e indirizzi
  - utilizza solo 32 parole riservate (27 K&R, 5 Ansi)
  - linguaggio strutturato
  - linguaggio per programmatori

Parole riservate

```
auto double int struct break else long switch
case enum register typedef char extern return
union const float short unsigned continue for
signed void default goto sizeof volatile do if
 static while
```

Generazione di codice eseguibile da un programma C



Librerie e compilazione separata

- Compilazione di un programma C
  - creazione del programma
  - compilazione del programma
  - collegamento del programma con le funzioni di libreria richieste

Compilazione di programmi C

• funzione	• Windows	• Linux
• Gestione progetto	• nmake.exe	• make
• Preprocessore	• cl.exe	• cpp
• Compilatore	• cl.exe	• gcc
• Linker	• link.exe	• ld

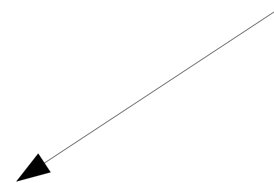
- In Linux:

```
$gcc [<opzioni>] file1.c file2.c file3.c  … [-l librerie]
```

- Normalmente:

```
$gcc file.c   #ompila e linka mettendo il codice eseguibile in a.out

$gcc file.c -c #compila e non linka mettendo il codice oggetto in file.o

$gcc file.c -o outfile  # compila e linka. codice exe in outfile

$gcc file.c -o outputfile -l libreria # compila e linka con libreria
```

Struttura di un programma C




```
Comandi del PREPROCESSORE

Prototipi di funzioni

Variabili globali

<tipo_ritorno> <nome_funz>(<elenco_argomenti>)

{

   <sequenza_istruzioni>

}
```

- Una delle funzioni è la funzione main() che definisce l'entry point del programma

Direttive del preprocessore

- Inclusione di file: `#include <file>`
  - `Per esempio #include <stdio.h>` ← stdio.h contiene definizioni e macro per IN/OUT
- Definizione di macro: `#define simbolo`
  - Esempio: `#define MACRO(x)  x * (x+5)`
  - Esempio: `#define inverti(x,y,temp) (temp)=(x);(x)=(y);(y)=(temp);`
- Compilazione condizionata: `#ifdef simbolo....#endif`
  - Esempio:

```
#define NAME "www.units.it"
…
#ifdef NAME
...istruzioni...
#endif
```

Struttura di un programma C

- La funzione <main> è l'unica obbligatoria, ed è definita come segue:

```
<tipo> main()
{
	[<dichiarazioni-e-definizioni>]
	[<sequenza-istruzioni>]
}
```

- Intuitivamente il main e' definito dalla parola chiave main()
- E' racchiuso tra parentesi graffe al cui interno troviamo
  - le dichiarazioni e definizioni
  - una sequenza di istruzioni

Struttura di un programma C

<dichiarazioni-e-definizioni>

- introducono i nomi di costanti, variabili, tipi definiti dall’utente

<sequenza-istruzioni>

- sequenza di frasi del linguaggio ognuna delle quali è un’istruzione
- Il main() è una particolare funzione

Linguaggio C

- Concetti elementari
  - dati (tipi primitivi, tipi di dato)
  - espressioni
  - dichiarazioni / definizioni
  - funzioni
  - istruzioni / blocchi

Tipi di dato primitivi

- caratteri
  - char
  - unsigned char
- interi con segno
  - short (int) -32768 ... 32767 (2byte=16 bit)
  - int [−2147483647, +2147483647] (4byte=32bit)
  - long (int) −9223372036854775807, +9223372036854775807 (8byte=64 bit)

- naturali (interi senza segno)
  - unsigned short (int) 0 ... 65535 (16 bit)
  - unsigned (int) 0 ... 4294967295 (32 bit)
  - unsigned long (int) 0 ... 1.8446744e+19 (64 bit)

Tipi di dato primitivi

- Reali
  - float singola precisione (32 bit)
  - double doppia precisione (64 bit)
- boolean
  - non esistono in C come tipo a sé stante
  - si usano gli interi:
    - zero indica FALSO
    - ogni altro valore indica VERO
    - convenzione: suggerito utilizzare uno per VERO

Costanti

- interi (in varie basi di rappresentazione)
  - decimale 12 70000 12L
  - ottale 014 0210560
  - esadecimale 0xFF 0x11170
- reali
  - in doppia precisione 24.0 2.4E1 240.0E-1
  - in singola precisione 24.0F 2.4E1F 240.0E-1F
- caratteri
  - singolo carattere racchiuso fra apici: ’A’ ’C’ ’6’
  - caratteri speciali: ’\n’ ’\t’ ’\’’ ’\\’ ’\"’

Stringhe

- Una stringa è una sequenza di caratteri delimitata da virgolette

  "ciao" "Hello\n"

- In C le stringhe sono semplici sequenze di caratteri di cui l’ultimo, sempre presente in modo implicito, è ’\0’

  "ciao" = {’c’, ’i’, ’a’, ’o’, ’\0’}

Variabili

- Una variabile è un’astrazione della cella di memoria.
- Formalmente, è un simbolo associato a un indirizzo fisico ...
- Perciò, l’ indirizzo di x è ad es. 1328 (fisso e immutabile!).

  Variabile x : locazione indirizzata da 1328 … che denota un valore .

  ..e il valore di x è attualmente ad es. 4 (può cambiare).

Caratteristiche di una variabile

- campo d’azione (scope): è la parte di programma in cui la variabile è nota e può essere manipolata
  - in C è determinabile staticamente
- tipo: specifica la classe di valori che la variabile può assumere (e quindi gli operatori applicabili)
- tempo di vita: è l’intervallo di tempo in cui rimane valida l’associazione simbolo/indirizzo fisico
- valore: è rappresentato (secondo la codifica adottata) nell’area di memoria associata alla variabile

Dichiarazione di variabile

- Una variabile utilizzata in un programma deve essere definita.
- La definizione è composta da
  – il tipo dei valori che possono essere assegnati alla variabile
  – il nome della variabile (identificatore)

    <tipo> <identificatore>;
- Esempi

```
int x; /* x deve denotare un valore intero */
float y; /* y deve denotare un valore reale */
char ch; /* ch deve denotare un carattere */
```

Definizione/Inizializzazione di una variabile

- Definizione e Inizializzazione di una variabile:

```
<tipo> <identificatore> = <espr> ;
```

- Esempio

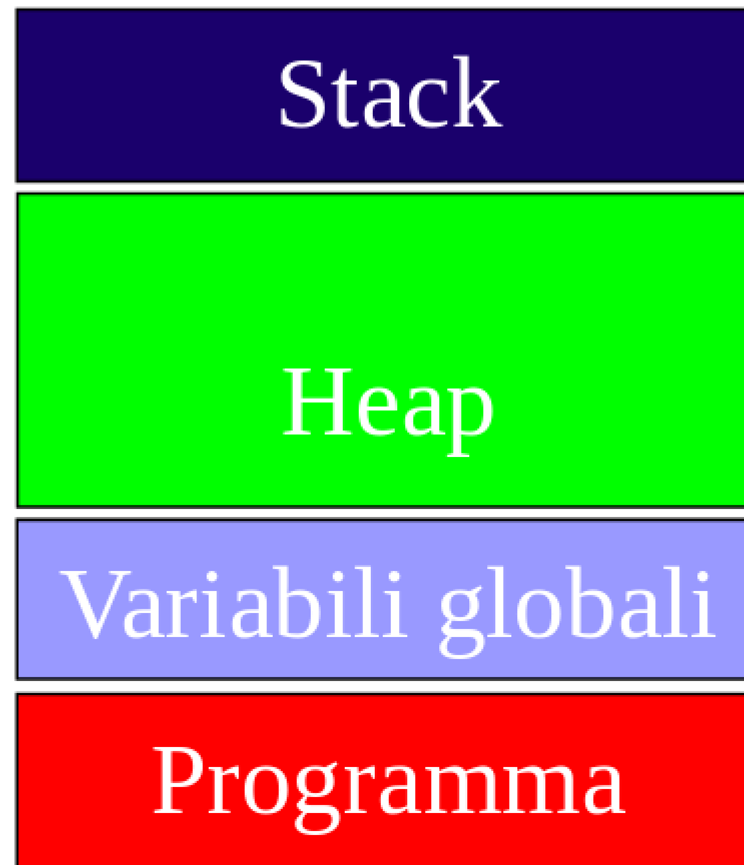
```
int x = 32;
double speed = 124.6;
```

- Conversione di tipo

```
(tipo) espressione
```

  - Esempio: `(char)65;  ← carattere con codice ASCII 65 in base 10 (A)`
    `(char)0x41;← carattere con codice ASCII 41 in base 16 (A)`
  - Esempio: `float x;`
    `x = (float) 7/5;`

Suddivisone concettuale della memoria per un programma C



Visibilità variabili

- Blocco di istruzioni: {...blocco...}
- Variabili locali: dichiarate dentro ad un blocco di istruzioni. Le variabili locali sono visibili solo dentro al blocco. Sono allocate sullo STACK.
- Variabili globali: dichiarate fuori da tutti i blocchi. Visibili da tutti i moduli dello stesso file.
- Dichiarazione extern: visibili tra diversi file
- Variabili formali: sono i parametri passati ad una funzione. Passati sullo STACK.
- Dichiarazione static. Una variabile statica viene allocata una sola volta. Visibile solo dentro il modulo dove e´ stata dichiarata.

Visibilità variabili

```c
/* file simple.c */
/* include macro e variabili */
#include <stdio.h>
#include <stdlib.h>

void func1(); /* definizione funzioni */

int count;    /* variabile globale */

int main(){   /* tipo d'uscita int */
    int i;    /* variabile locale */
    for(i=0; i<10; i++) {
        count=i*2;
        func1();
    }
    return 0;
}

void func1(){
    printf("count=%d\n",count);
}
```

```c
/* file simple1.c*/
/* include macro e variabili */
#include <stdio.h>
#include <stdlib.h>

void func1(int); /* definizione funzioni */

int main(){
    int i;     /* variabile locale */
    int count; /* variabile globale */
    for(i=0; i<10; i++) {
        count=i*2;
        func1(count);
    }
    return 0;
}

void func1(int c){ /* c variabile formale */
    printf("count=%d\n",c);
}
```

Compilazione/esecuzione:
$gcc simple.c -o simple
$./simple

Compilazione/esecuzione:
$gcc simple1.c -o simple1
$./simple1

Visibilità variabili

```
/* file simple2.c */
/* include macro e variabili */
#include <stdio.h>
#include <stdlib.h>

void func1(); /* definizione funzioni */
void func2();

int count;    /* variabile globale */

int main(){
    int i;


    for(i=0; i<10; i++) {
    count=i*2;
    func1();
    }
    return 0;
}
```

```
/* file func.c */
#include <stdio.h>

extern count;

void func1(); /* definizione funzioni */
void func2();

void func1(){
    func2();
    printf("count=%d\n",count);

}

void func2(){
    int count; /* variabile locale */
    for(count=0; count<3; count++)
    printf(".");
}
```

Compilazione/esecuzione:
$gcc simple2.c func.c -o simple2
$./simple2

Espressioni

- Il C e' un linguaggio basato su espressioni
- Un'espressione e' una notazione che denota un valore mediante un processo di valutazione
- Una espressione puo' essere semplice o composta tramite aggregazione di altre espressioni

Valutazione di un'espressione

- Una variabile
  – può comparire in una espressione
  – può assumere un valore dato dalla
- valutazione di un'espressione
  – double speed = 124.6;
  – double time = 71.6;
  – double km = speed * time;

Classificazione degli operatori

- Due criteri per classificare gli operatori:
  - in base al tipo degli operandi
  - in base al numero di operatori

In base al tipo degli operandi	In base al numero degli operandi
Aritmetici	Unari
Relazionali	Binari
Logici	Ternari

Operatori aritmetici

operazione	**operatore**	**in C**
inversione di segno	unario	-
somma	binario	+
differenza	binario	-
moltiplicazione	binario	*
divisione fra interi	binario	/
divisione fra reali	binario	/
modulo (fra interi)	binario	%

NB:la divisione a/b è fra interi se sia a sia b sono interi,

è fra reali in tutti gli altri casi

Operatori di relazione

uguaglianza	==
diversità	!=
maggiore di	>
minore di	<
maggiore o uguale a	>=
minore o uguale a	<=

Espressioni e operatori di relazione

- In C non esiste il tipo boolean
- In C le espressioni relazionali denotano un tipo intero

  - 0 denota il valore falso (condizione non verificata)

  - 1 denota il valore vero (condizione verificata)

- Quindi sono possibili espressioni miste come

  (n != 0) == n

    da evitare

Operatori logici

<table>
<tr><th>connettivo logico</th><th>operatore</th><th>in C</th></tr>
<tr><td>not (negazione)</td><td>unario</td><td>!</td></tr>
<tr><td>and</td><td>binario</td><td>&&</td></tr>
<tr><td>or</td><td>binario</td><td>||</td></tr>
</table>

Espressioni e operatori logici

- In C le espressioni logiche denotano un tipo intero da interpretare come vero (1) o falso (0)
- Anche qui sono possibili espressioni miste, utili in alcuni casi,

  5 && 7   !5   0 || 33

- Valutazione in corto circuito
  - la valutazione dell’espressione cessa appena si e' in grado di determinare il risultato
  - il secondo operando e' valutato solo se necessario

Valutazione in corto circuito

- if(22 || x) e' vera in partenza perché 22 e' vero
- If(0 && x) e' falsa in partenza perché 0 e' falso
- if(a && b && c) il secondo && viene valutato solo se a && b e' vero
- if(a || b || c) il secondo || viene valutato solo se a || b è falso

Espressioni condizionali

- Una espressione condizionale è introdotta dall’operatore ternario

  condiz ? espr1 : espr2

- L’espressione denota o il valore denotato da espr1 o quello denotato da espr2 in base al valore della espressione condiz
- Se condiz è vera, l’espressione nel suo complesso denota il valore denotato da espr1
- Se condiz è falsa l’espressione nel suo complesso denota il valore denotato da espr2

  3 ? 10 : 20 denota sempre 10 (3 è sempre vera)

  x ? 10 : 20 denota 10 se x è vera (diversa da 0),

  oppure 20 se x è falsa (uguale a 0)

  (x>y) ? x : y denota il maggiore fra x e y

Esempio

```
#include <stdio.h>
int main()
{
    int a; /*primo valore a*/
    int b; /*secondo valore*/
    scanf(“%d”,&a);  /*lettura da standard input*/
    scanf(“%d”,&b);
    if (a>b) {
          printf(“%d”,a); /*scrittura su standard output*/
          printf(“%d”,b);
     }
     else{
          printf(“%d”,b);
          printf(“%d”,a);
     };
}
```

Esempio

```
/* un menu */
#include <stdio.h>
void prima(); void seconda(); void terza();
int main()
{
   char car;
   printf("1 per prima scelta\n");
   printf("2 per seconda scenta\n");
   printf("3 per terza scelta\n");
   do {
      printf("inserisci la tua scelta\n");
      car = getchar(); /*legge un car. dallo standard in*/
      switch(car)      /* switch() e' il case del C*/
      {
        case '1': prima(); break;
        case '2': seconda(); break;
        case '3': terza(); break;
      }
    }while(car=='1' || car=='2' || car=='3');
}
```

Esempio

```
/* media dei numeri pari presenti in una sequenza di N numeri*/
#include <stdio.h>
int main()
{
   int N, i, numero, somma;
   float media;
   somma = i = 0;
   scanf("%d", &N);
   while (N != 0)
   {
      scanf("%d", &numero);
      N = N -1;
      if (numero%2) continue;
      somma = somma + numero;
      i = i +1;
    }
    media = ((float)somma)/i;
    printf("Somma dei numeri pari: %d Media: %f\n", somma,media);
}
```

Tipi di dato strutturato: array

- Array: int A[10]; float B[10];
- Inizializzazione
  - int v[4] = {2, 7, 9, 10}
  - int v[ ] = {2, 7, 9, 10}
- Il C non effettua controllo sui limiti degli array

Esempio

```
#define DIM 4
#include <stdio.h>
main()
{
    int v[DIM];
    int i, max=0;
    for (i =0 ; i<DIM; i++)
          scanf("%d", &v[i]); /*leggi il vettore*/
    for (max = v[0], i =1 ; i<DIM; i++)
          if (v[i]>max) max=v[i];
    printf("%d", max);
}
```

Tipi di dato strutturato

- Array multidimensionali: int matrice [N][M]
- Esempio:

```
#include <stdio.h>
main()
{
    float m[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},
                   {13,14,15,16}};
    float somma=0;
    int i,j;
    for(i=0;i<4;i++)
       for(j=0;j<4;j++)
           somma+=m[i][j];
    printf("media=%f\n", somma/4*4);
}
```

Tipi di dato strutturato

- Stringhe di caratteri
  - Array di caratteri terminato dal carattere '\0'.
  - L'array di N caratteri può ospitare stringhe lunghe al più
    N-1 caratteri, perchè una cella è destinata al terminatore '\0'

- Una stringa di caratteri si può inizializzare, come ogni altro array, elencando le singole componenti:
    - char s[4] = {'a', 'p', 'e', '\0'};
  - Oppure anche, più brevemente, con la forma compatta seguente:
    - char s[4] = “ape”;

- Il carattere di terminazione '\0' è automaticamente incluso in fondo. Quindi, ATTENZIONE ALLA LUNGHEZZA!

Lunghezza di una stringa

```
#include <stdio.h>
main ()
{
   char s[] = “Stringa di prova”;
   int lung;

    for (lung=0 ; s[lung] != '\0'; lung++);
   printf (“lunghezza della stringa “%d”, lung);
}
```

Copia una stringa

```
#include <stdio.h>
#define N 6
\\ copia N caratteri
main ()
{
    char s[] = “Stringa di prova”;
    char s2[N]; int i;
    for (i = 0; s[i] != “\0” && i<N-1; i++)
          s2[i] = s[i]; s2[i]= “\0”;
}
```

Precedenza di stringhe

se s1 precede s2 → stato negativo
se s2 precede s1 → stato positivo
se s1 e uguale a s2 → stato nullo

```
#include <stdio.h>
main ()
{
      char s1[] = "amaca"; char s2[] = "amace";
      int stato,i;
      for ( i=0; s1[i] != "\0" && s2[i] != "\0" && s1[i] == s2[i]; i++){
         stato= s1[i]-s2[i];
         printf("s1[i]=%c,s2[i]=%c,stato=%d\n",s1[i],s2[i],stato);
      }
      stato= s1[i]-s2[i];
      printf("stato=%d\n",stato);
}
```

Per raccogliere queste semplici operazioni → Libreria sulle stringhe

- Il C fornisce una libreria per operare sulle stringhe:

  #include <string.h>

- Include funzioni per:
  - Copiare una stringa in un'altra (strcpy)
  - Concatenare due stringhe (strcat)
  - Confrontare due stringhe (strcmp)
  - Cercare un carattere in una stringa (strchr)
  - Cercare una stringa in un'altra (strstr)

Tipi di dato strutturato

- **struct**: collezione finita di variabili non necessariamente dello stesso tipo ognuna identificata da un nome.

```
struct <nome struttura}{
    <definizione di avriabile>
} <eventuali variabili
```

- Esempio:

```
struct persona{
   char nome[20];
   int eta;
   float stipendio;
}pers;          /*variabile struttura*/
struct persona pers1,pers2;
```

- Esempio:

```
struct complesso {float pr, pi;}   /*occupa 8 byte*/
```

Tipi di dato strutturato

- Una volta definita la struttura, I campi vengono riferiti con la **notazione punto**
  - Esempio: `p1.x=10; p1.y=10; p2.x=5; p2.y=1;`
  - Esempio: `struct punto{float x,y;}p1={10,10}; punto p2={5,1};`
- E' possibile:
  - Assegnare una struttura ad un'altra: `punto p=p1;`
  - Una funzione restituisce una struttura
  - Passare una struttura ad una funzione
- Esempio:

```
struct orografia{
     char nome[20]; /*20 byte*/
     int longit;    /*4 byte*/
     int latid;     /*4 byte*/
     int altezza;   /*4 byte*/
};
struct orografia montebianco;
struct orografia monti[10];

monti[0].altezza=4800;
```

→ Spazio occupato totale=32 byte

→ Spazio occupato totale=320 byte

Tipi di dato strutturato

- Istruzione typedef
  - Il C permette di definire esplicitamente nomi nuovi per i tipi di dati, tramite la parola chiave typedef
  - L'uso di typedef consente di rendere il codice più leggibile.
- Formato: `typedef tipo nuovo_nome_tipo ;`
- Esempio

```
typedef struct{float pr,pi;}complesso;
complesso N1,N2;
```

Tipi di dato strutturati

```
#include<stdio.h>

struct complex{
  int real;
  int img;
};

typedef struct complex comp;

comp add(comp,comp);
void show(comp);

int main(){
  comp c1,c2,c3;
  c1.real=1; c1.img=2;
  c2.real=3; c2.img=4;
  c3=add(c1,c2);
  show(c3);
  return 0;
}

void show(comp c){
  printf("%d + i%d\n",c.real,c.img);
}
com add(comp c1,comp c2){
  com c3;
  c3.real=c1.real+c2.real;
  c3.img=c1.img+c2.img;
  return c3;
}
```

Funzioni

- Esempi:

```
float f (){ return (2 + 3 * sin(0.75)); }

float f1 ( int x ){return(2 + x * sin(0.75)); }

void f2() {printf(“stringa”);}
```

- La lista degli argomenti può essere vuota
- Definizione dei prototipi di funzione:
  - Descrivono la proprietà della funzione senza definirne la realizzazione.
  - Anticipano le caratteristiche di una funzione definita successivamente.
  - <tipo> <nome_funzione> (<tipo argomenti>) ;
- Attenzione: se le funzioni sono definite prima del main, non sono necessari i prototipi

Funzioni

- Passaggio dei parametri per valore
- Esempio:

```
#include <stdio.h>
int massimo (int a, int b)  /*calcola il massimo fra a e b) */
{
    if (a > b) return a;
    else return b;
}
int sommamax (int a1, int a2, int a3, int a4)
{
    return (massimo(a1, a2) + massimo(a3, a4));
}
main()
{
    int A=1, B=3, C=5, D=4;

    printf(“%d\n”, sommamax(A,B,C,D));
}
```

- Array, strutture passati come parametri di una funzione

Funzioni

- Array passati come parametri. Esempio:

```
#include <stdio.h>
void lunghezza (char W[]);
void main()
{
     char V[10];
      ….
     lunghezza (V)
      …..
 }

 int lunghezza(char s[])
 {
   int lung = 0;
   for (;s[lung]!='\0';lung++);
   return lung;
 }
```

Funzioni

```
/* Visualizza le potenze dei
   numeri compresi fra 1 e 10 */
#include <stdio.h>
#include <stdlib.h>
#define N 10
#define M 4
int potenza(int a, int b);
void tabella (int p[M][N]);
void stampa_tabella (int p[M][N]);
void main ()
{
    int p[M][N];
    tabella(p);
    stampa_tabella(p);
}
int potenza(int a, int b)
{
    int t=1;
    for ( ; b; b--) t = t*a;
    return t;
}
```

```
void tabella (int p[M][N])
{
     int i, j;
     for (j = 0; j<N; j++)
        for (i = 0; i<M; i++)
           p[i][j] = potenza(j+1, i+1);
}
void stampa_tabella (int p[M][N])
{
      int i, j;
      for (i = 0; i<M; i++){
         for (j = 0; j<N; j++)
            printf(“%10d”, p[i][j] );
            printf (“\n”);
         }
}
```

Puntatori

- L’uso dei puntatori è una importantissima differenza tra Java e C.
- In C ogni variabile caratterizzata da due valori: un indirizzo della locazione di memoria in cui è allocat la variabile, ed il valore contenuto in quella locazione di memoria, cioè il valore della variabile.
- Un puntatore è un tipo di dato, è una variabile che contiene l’indirizzo in memoria di un’altra variabile, cioè un numero che indica in quale cella di memoria comincia la variabile puntata:

puntatore → *variabile*

- Per dichiarare un puntatore p ad una variabile di tipo tipo, l’istruzione e’:

```
tipo *p;
```

- L’ operatore & fornisce l’indirizzo di una variabile, perciò l’ istruzione

```
p = &c
```

scrive nella variabile p l’indirizzo della variabile c, ovvero:

```
tipo c, *p; //dichiaro una var c di tipo tipo ed
            //un puntatore p a tipo
p = & c ;   //assegno a p l’indirizzo di c
```

Puntatori

- L’operatore * viene detto operatore di indirezione o deriferimento.
- Quando una variabile di tipo puntatore è preceduta dall’operatore * , indica che stiamo accedendo all’oggetto puntato dal puntatore.
- Quindi con *p indichiamo la variabile puntata dal puntatore.

```
int c, *p; //dichiaro una var ed un puntatore p a int
p = &c ;   //assegno a p l’indirizzo di c c
C = 5;     //assegno a c il valore 5
printf("%d\n", *p); //stampo il valore puntato da p.
// viene stampato 5
```

- Consideriamo gli effetti delle seguenti istruzioni:

```
int *pointer; /* dichiara pointer come puntatore a int */
int x=1,y=2;
pointer= &x; /* assegna a pointer l’indirizzo di x */
y=*pointer;  /* y = il contenuto dell’int puntato da pointer*/
x=pointer    /* assegna ad x l’indirizzo contenuto in pointer*/
*pointer=3;  /* assegna 3 alla variabile puntata da pointer */
```

Puntatori

- Quindi con pointer possiamo considerare tre possibili valori:
  - 1) pointer → contenuto o valore della variabile pointer cio e ́ l’indirizzo della locazione di memoria a cui punta
  - 2) &pointer → indirizzo fisico della locazione di memoria del puntatore
  - 3) *pointer → contenuto della locazione di memoria a cui punta
- Attenzione.

```
int *ip;
*ip=100;
```

  è un grave errore: la locazione dove scrivo il valore 100 DEVE ESSERE ALLOCATA! Altrimenti potrebbe cadere in una zona importante del sistema!

- Come allocare spazio? o puntando ad una variabile allocata o usando

```
void *malloc(int nr_byte)
```

Esempio malloc

```
int *ip;
ip=malloc(4);   /*4 byte in memoria heap*/
*ip=83;
```

Aritmetica dei puntatori

- Operazioni semplici +, - , ++ e -- sugli indirizzi
- Ma: il risultato numerico di un’operazione aritmetica su un puntatore dipende dal tipo di dato a cui il puntatore punta.
- Se p un puntatore di tipo puntatore a char (char *p) l’istruzione p++ aumenta effettivamente di uno il valore del puntatore p, che punterà al successivo byte.
- Invece se p è un puntatore a short int (short int *p) l’istruzione p++ incrementa di 2 il valore del puntatore p, che punterà allo short int successivo
- se il puntatore punta a void (void *p) il puntatore viene incrementato o decrementato a passi di un byte.

Puntatori e array

- Il nome di un array e´ un puntatore alla prima locazione dell'array!
- Esempio: Se ho

```
int buf[100];
```

`buf[1] e *(buf+1)` sono due modi EQUIVALENTI per accedere al 2° elemento di `buf, ovvero al 33-esimo byte`

- Modo veloce per copiare l'array buf nell'array arr:

```
int buf[100]; int arr[100]; while(*(arr++) = *(buf++));
```

- Esempio

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

int main(void)
{
    short i,j;
    float res;
    int arr[16];
    int buf[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
    int *pa=arr, *pb=buf;

    while(*(pa++)=*(pb++));
     for(i=0;i<16;i++)printf("%d ",arr[i]);
    printf("\n");
}
```

Puntatori e array

- Attenzione: un array bidimensionale [M×N] costruito dinamicamente è un array di M puntatori che puntano a array monodimensionali di N elementi.

- Se l'array è costruito staticamente, esempio float M[3][4], l'array è una successione di elementi. In questo caso accedere all'elemento `[i][j]` vuol dire

```
float *p=&M[0][0]; elem_i,j=*(p+i*N+j)
```

Puntatori e array

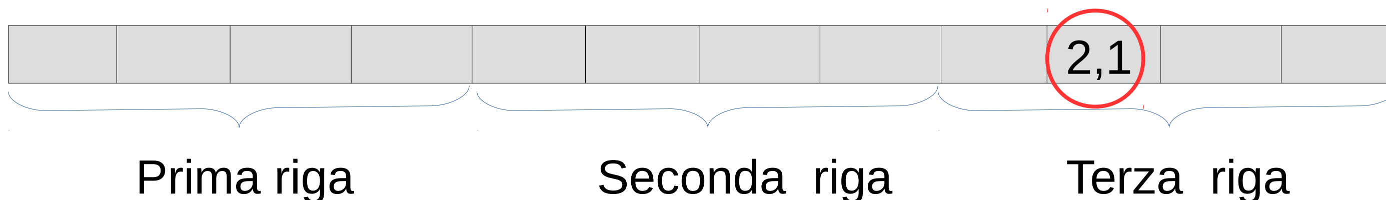
int A[M][N]=int A[3][4];

int *p=&A[0][0];

p=&A[0][0]

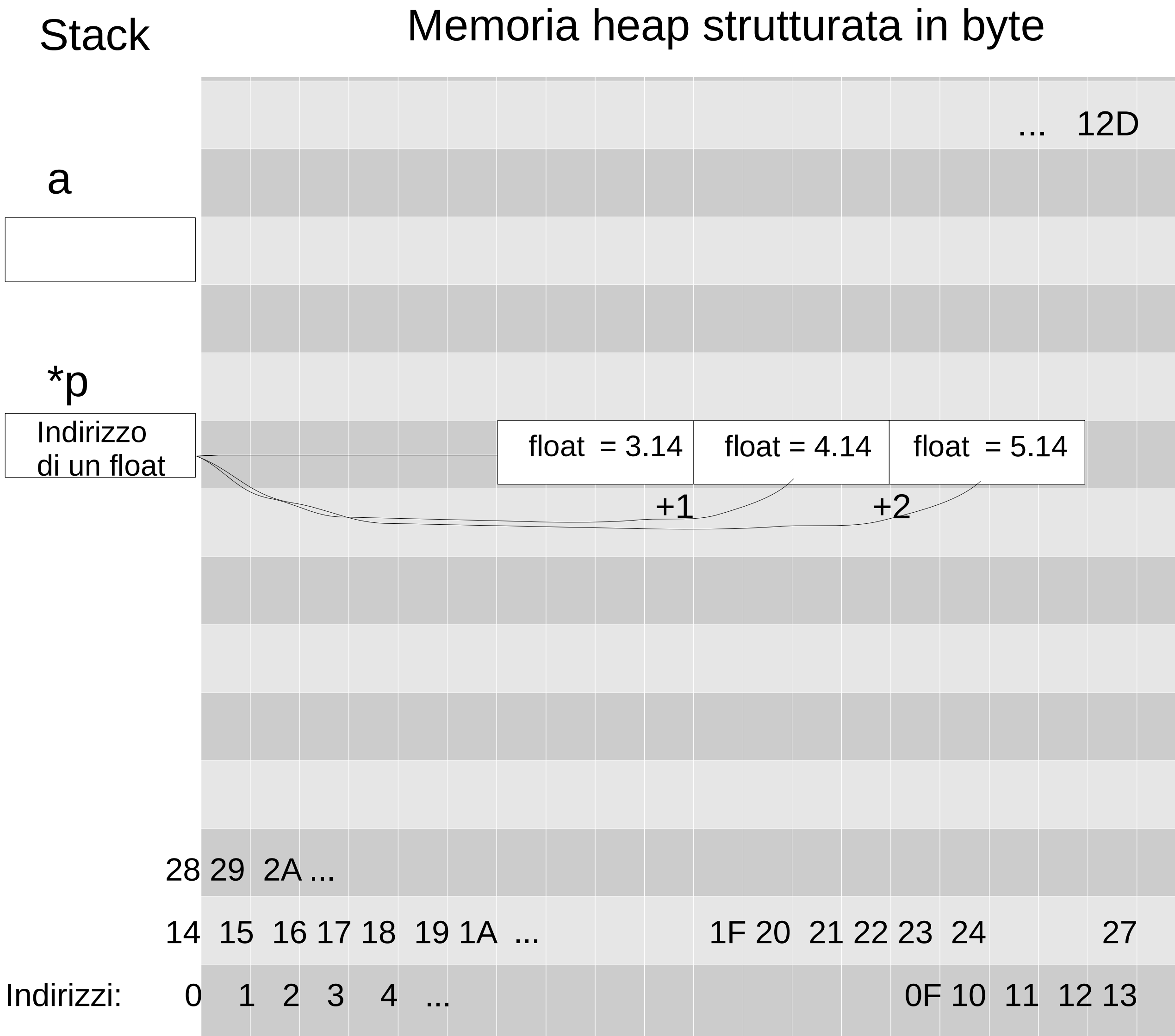
A[2][1] = `*(p+2*4+1) → *(p+i*N+j)`

Una matrice bidimensionale può essere allocata unidimensionamente per righe:



Prima riga Seconda riga Terza riga

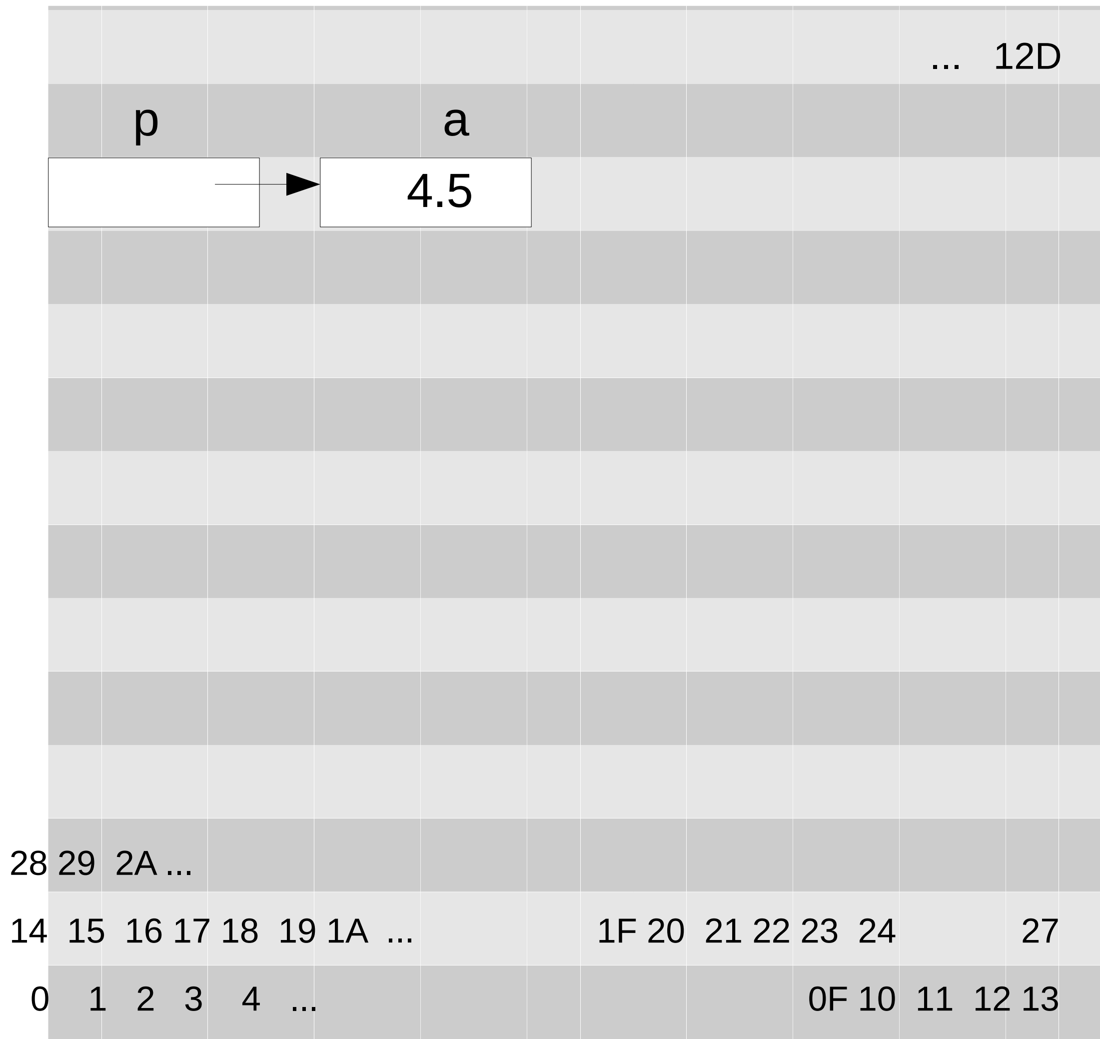
Salviamo dei valori float in un array di float in heap:




```
void main{
    float a=4.5; //stack
    float *p;

    p=malloc(3*sizeof(float));

    *p=3.14;
    *(p+1)=4.14;
    *(p+2)=5.14;

  //In alternativa:

    p[0]=3.14;
    p[1]=4.14;
    p[2]=5.14;
}
```

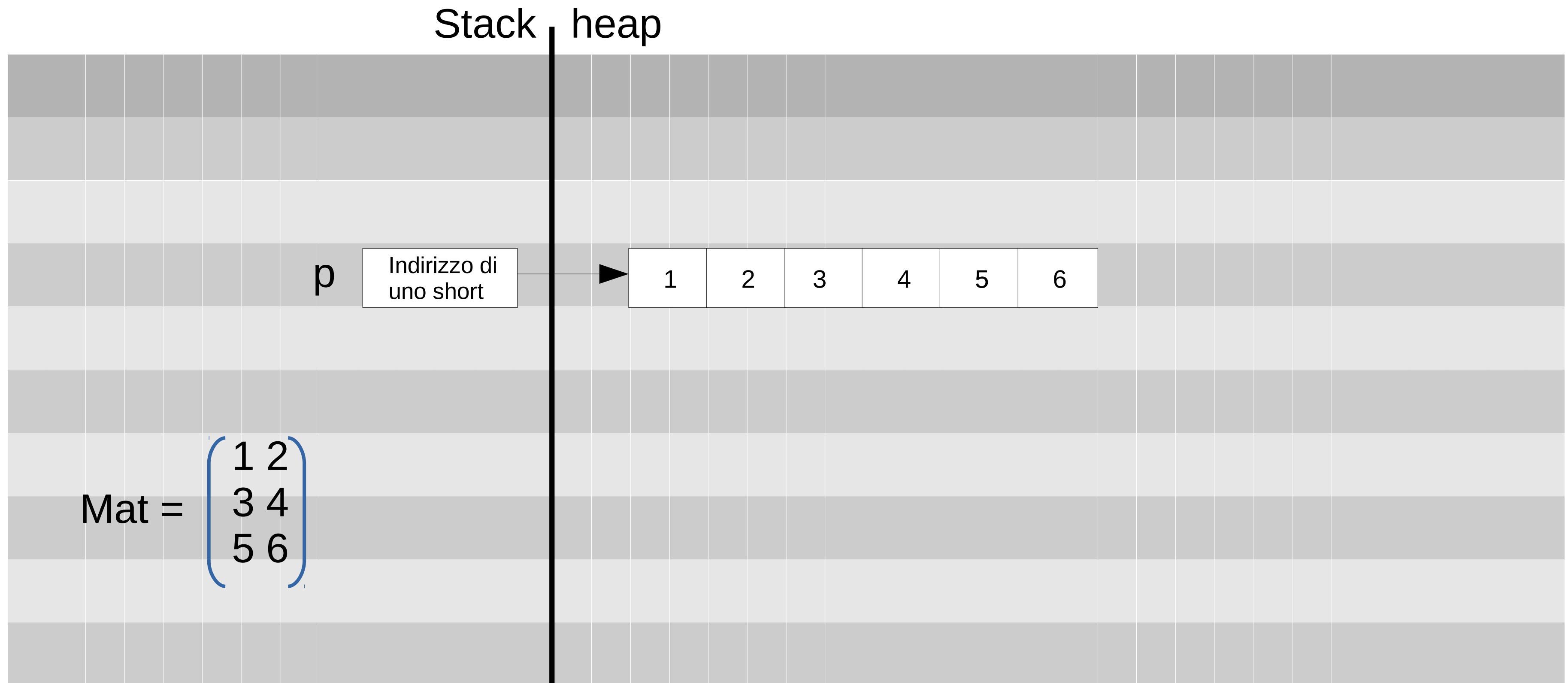
***p è un array p[ ] !!**

Memoria stack




```
#include <stdio.h>


int main(void)
{
        float a=4.5;
        float *p;

        p =&a;

        printf("%f allocato in %d\n",a,p);
        printf("%f allocato in %d\n", *p, p);
        printf("%f allocato in %d\n", a,&a);
}
```

Memorizziamo una matrice NxM in un array monodimensionale in heap:

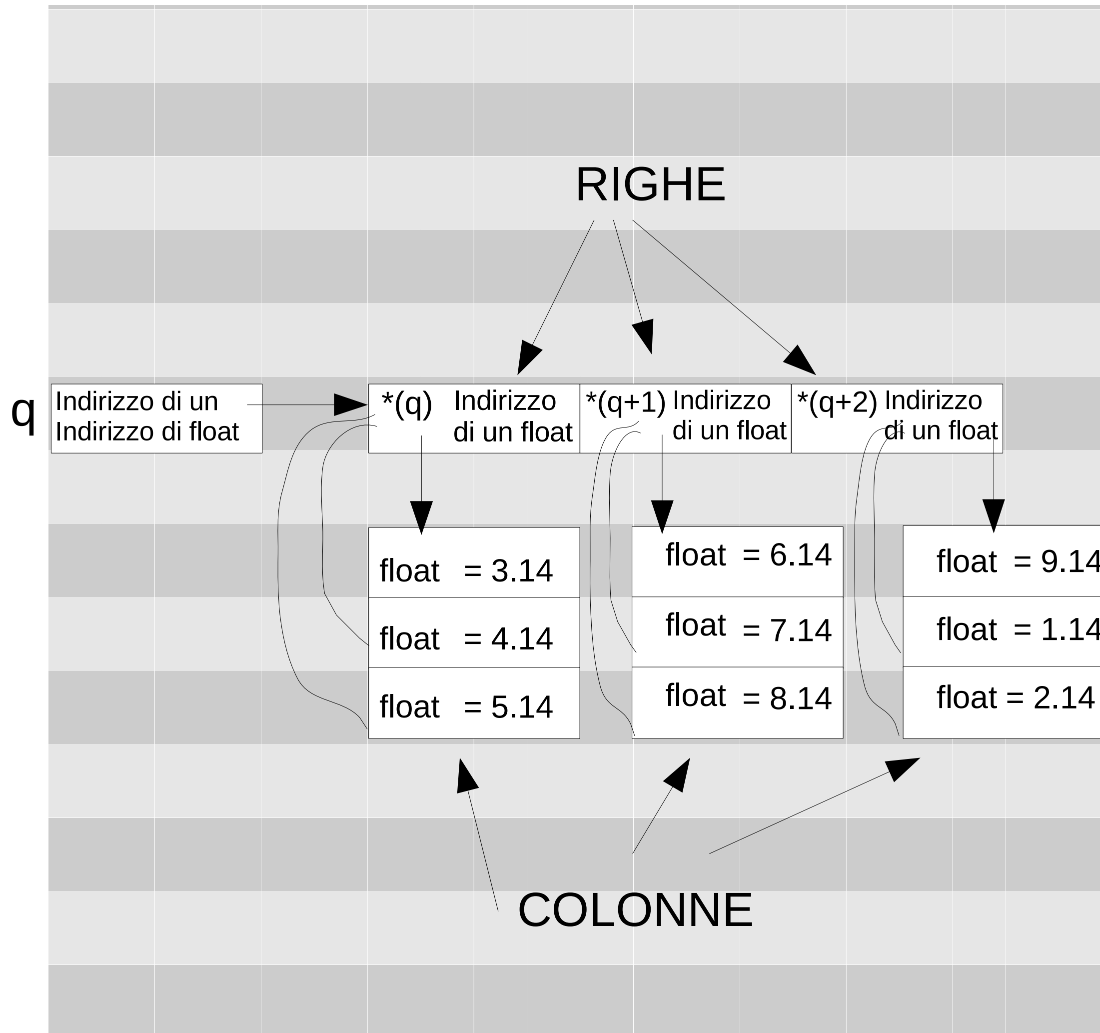



```
short Mat[3][2]={{1,2},{3,4},{5,6}};
short *p=malloc(3*2*sizeof(short));
*(p+0) = 1;
*(p+1) = 2;
*(p+2) = 3;
*(p+3) = 4;
*(p+4) = 5;
*(p+5) = 6;
```

algoritmo →

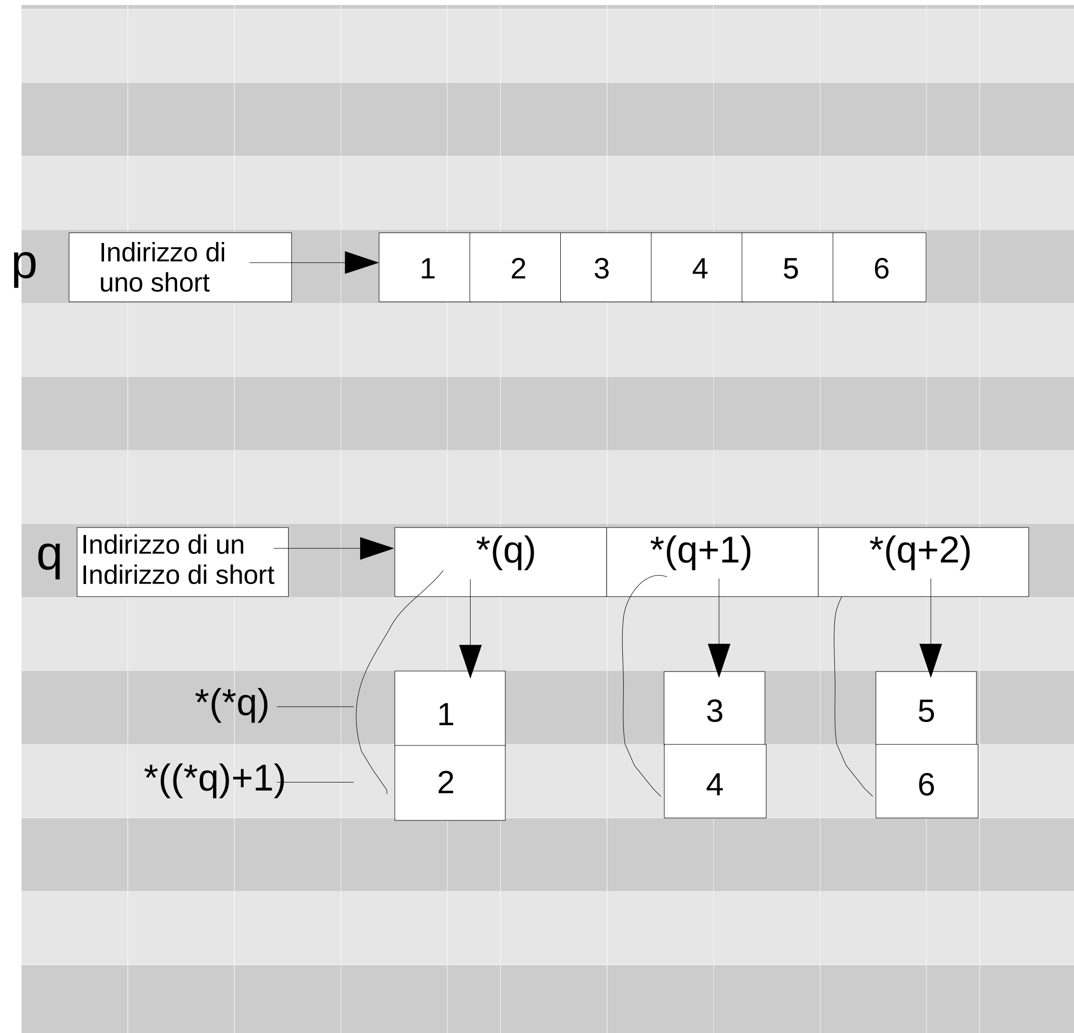
```
for(i=0;i<3;i++)
    for(j=0;j<2;j++)
        *(p+i*2+j)=Mat[i][j];
```

Salviamo dei valori float in un array Bidimensionale 3x3 di float in heap:




```
float **q;
q=malloc(3*sizeof(float*));
*q=malloc(3*sizeof(float));
*(q+1)=malloc(3*sizeof(float));
*(q+2)=malloc(3*sizeof(float));
*(*(q))=3.14;
*(*(q)+1)=4.14;
*(*(q)+2)=5.14;
*(*(q+1))=6.14;
*(*(q+1)+1)=7.14;
*(*(q+1)+2)=8.14;
*(*(q+2))=9.14;
*(*(q+2)+1)=1.14;
*(*(q+2)+2)=2.14;
```

Copia di una matrice 3x2 memorizzata in un array monodimensionale di short in un array Bidimensionale di short in heap:



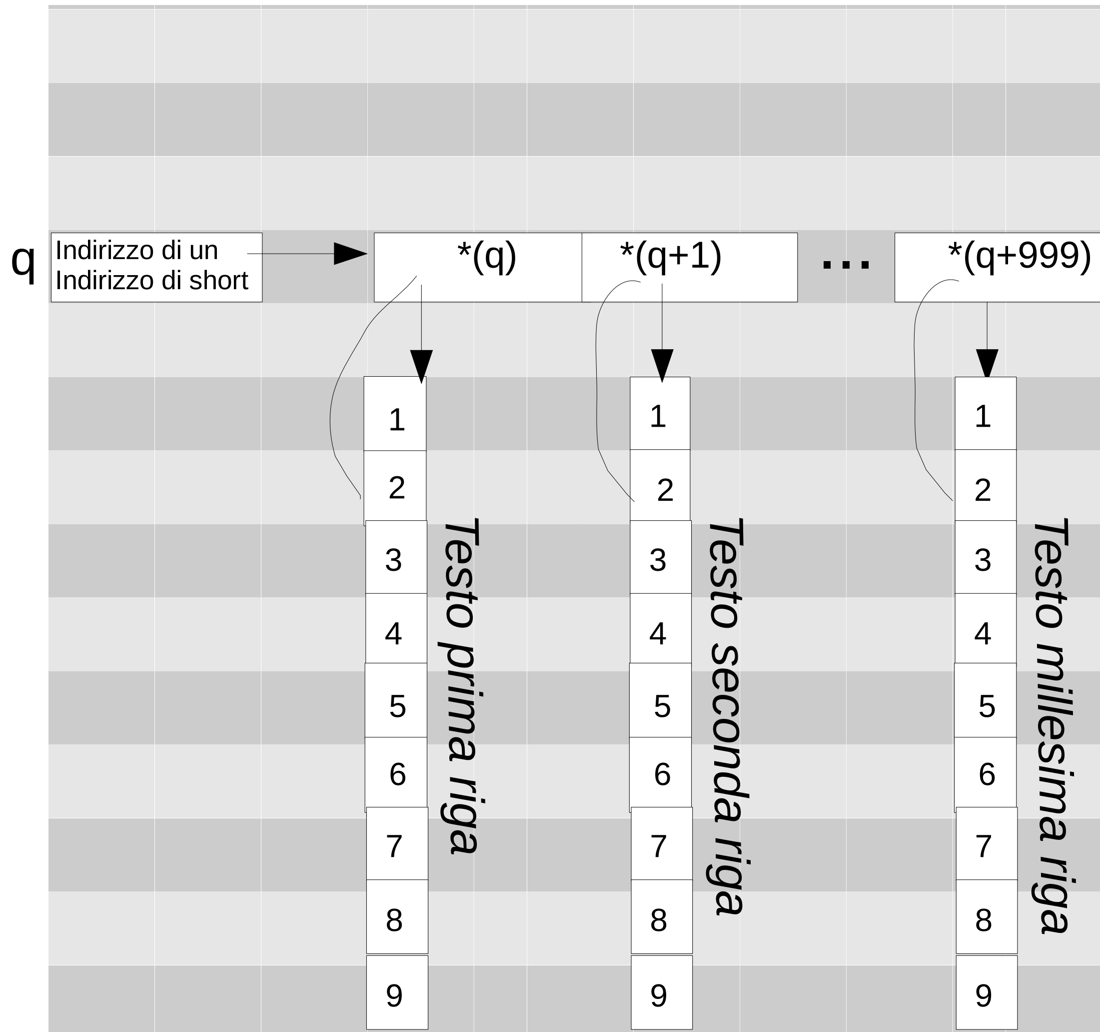

```
short **q;
q=malloc(3*sizeof(short*));
*(q)=malloc(2*sizeof(short));
*(q+1)=malloc(2*sizeof(short));
*(q+2)=malloc(2*sizeof(short));
*(*q)=*(p+0*2+0);
*((*q)+1)=*(p+0*2+1);
*(*(q+1))=*(p+1*2+0);
*(*(q+1)+1)=*(p+1*2+1);
*(*(q+2))=*(p+2*2+0);
*(*(q+2)+1)=*(p+2*2+1);
```

```
short *p;
p=malloc(6*sizeof(short);
/* inizializza I valori */
```

Algoritmo:

```
short **q=malloc(3*sizeof(short*));
short *p=malloc(3*2*sizeof(short));
for(i=0;i<3;i++){
     q[i]=malloc(2*sizeof(short));
     for(j=0;j<2;j++)
          *(*(q+i)+j)=*(p+i*2+j);
}
```

Legge per righe un file di testo e le salva un array Bidimensionale di stringhe in heap:



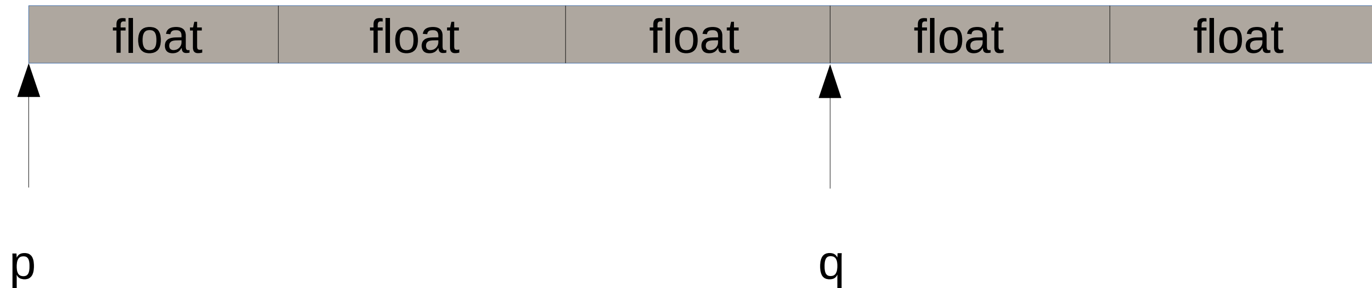

```
char **q;
q=malloc(1000*sizeof(char*));
fp=fopen("testo.txt","r"));
fgets(testo, size, fp);
*(q)=malloc(strlen(testo));
strcpy(*(q),testo);
fgets(testo, size, fp);
*(q+1)=malloc(strlen(testo));
strcpy(*(q+1),testo);
...
fgets(testo, size, fp);
*(q+999)=malloc(strlen(testo));
strcpy(*(q+999),testo);
```

Algoritmo:

```
char **data=malloc(1000*sizeof(char*));
fp=fopen("testo.txt","r"));}
for(i=0;i<10;i++){
    fgets(riga, size, fp);
    data[i]=malloc(strlen(riga));
    strcpy(data[i],riga);
}
close (fp);
```

Quanti elementi ci sono tra due puntatori?

float *p, *q;



I puntatori sono l'indirizzo del byte puntato cioè sono interi.
La differenza tra puntatori è il numero di byte compreso:
Il numero di byte compreso tra q e p è (q - p)

Quanti elementi sono compresi? Dipende dal tipo di dato.
Se sono float: numero elementi =

$$\frac{(q-p)}{sizeof(float)}$$

In generale:

$$\frac{(q-p)}{sizeof(type)}$$

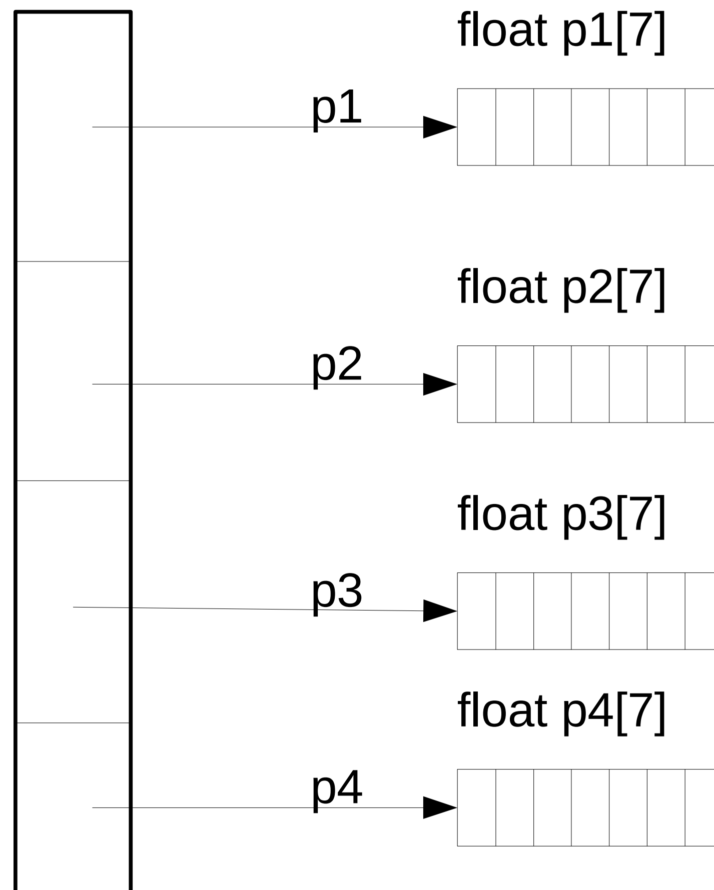
Programma per scrivere una matrice bidimensionale 4X4 in Stack

```
##include <stdio.h>
#include <math.h>
int funct2(float *a)
{
      short i,j;
      i=0;j=1;printf("elem %d %d=%f\n", i,j, *(a+i*4+j) );
      i=1;j=1;printf("elem %d %d=%f\n", i,j, *(a+i*4+j));
      i=2;j=2;printf("elem %d %d=%f\n", i,j, *(a+i*4+j));
      i=3;j=3;printf("elem %d %d=%f\n", i,j, *(a+i*4+j));
      return 0;
}
int main(void)
{
          short i,j;
          float res;
          float mat[4][4]={1,2,3,4,
                           5,6,7,8,
                           9,10,11,12,
                           13,14,15,16};
          float *p=&mat[0][0];

          funct2(p);
          printf("fine\n");
}
```

Programma per creare una matrice bidimensionale in Stack

Array di 4 indirizzi:

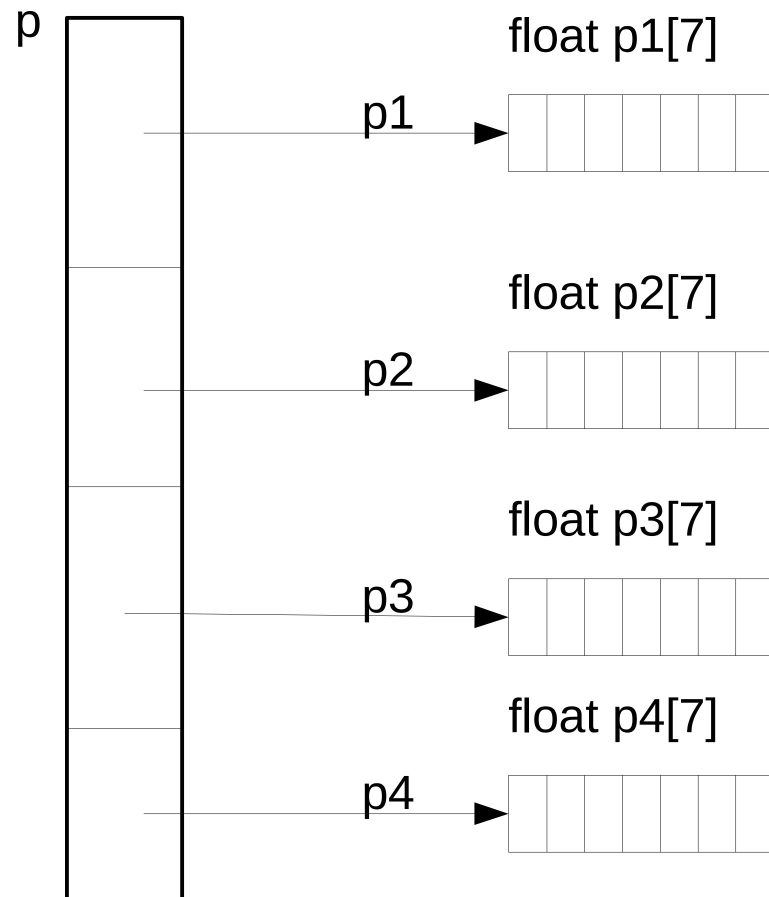
```
→ float *pp[4]; float **p=pp, p1[7]; p2[7]; p3[7]; p4[7];
```



Programma per creare una matrice bidimensionale in Stack

- Quindi la matrice è un puntatore di puntatori, definito come **p
- L’elemento i,j può essere acceduto in uno dei seguenti modi:

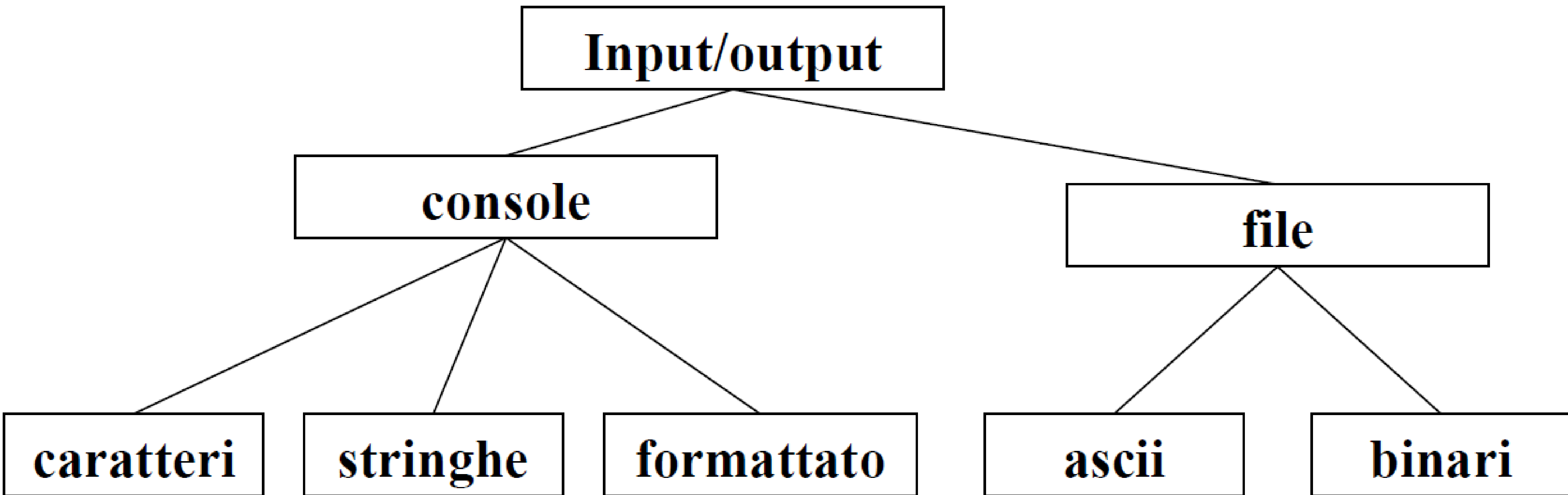
```
p[i][j]
*(p[i]+j)
(*(p+i))[j]
*(*(p+i)+j)
*(&p[0][0] + M*i + j)
```



Programma per creare una matrice bidimensionale in Stack

```
#include <stdio.h>
float funct2(float **a)
{
      short i,j;
      i=3,j=1; printf("elem 3,1=%f\n", *(*(a+i)+j) );
      i=1;j=2; printf("elem 1,2=%f\n", *(a[i]+j));
      i=3;j=3; printf("elem 3,3=%f\n", a[i][j] );
      i=2;j=3; printf("elem 2,3=%f\n", *(&a[0][0] +7*i + j));
      return 0;
}

int main(void)
{
      short i,j;
       /*inizializzazione*/
      float p1[7]={1,2,3,4,5,6,7};
      float p2[7]={8,9,10,11,12,13,14};
      float p3[7]={15,16,17,18,19,20,21};
      float p4[7]={22,23,24,25,26,27,28};
      float *pp[4], **p=pp;;
      pp[0]=p1; pp[1]=p2;      /*caricamento vettore di puntatori*/
      pp[2]=p3; pp[3]=p4;
      funct2(p);
      printf("fine\n");
}
```

Input/Output



La libreria standard del C

- Per usare la libreria standard, non occorre inserirla esplicitamente
- Ogni file sorgente che ne faccia uso deve includere gli header opportuni che contengono le dichiarazioni necessarie
- Header della Libreria standard
  - input-output stdio.h
  - funzioni matematiche math.h
  - gestione di stringhe string.h
  - operazioni su caratteri ctype.h
  - gestione dinamica della memoria stdlib.h

Header stdio.h

- l’input avviene di norma dal canale standard di input (stdin)
- l’output avviene di norma dal canale standard di output (stdout)
- input e output avvengono sotto forma di una sequenza di caratteri
- tale sequenza è terminata dal carattere speciale EOF (end of file), che varia da una piattaforma ad un’altra
- di norma il canale standard stdin coincide con la tastiera
- di norma il canale standard stdout coincide con il video
- esiste inoltre un altro canale di output, riservato ai messaggi di errore: stderr

Header stdio.h

- Due operazioni base
  - scrivere un carattere su un canale di output

```
putchar(ch); /*scrive un carattere sul canale di output*/
```

  - leggere un carattere dal canale si input

```
ch = getchar(); /*effettua l'echo e attende invio*/
```

- Lettura di stringhe dallo standard input

```
char *gets( char *s );/* la stringa è memorizzata nella
                         locazione indicata nel puntatore s */
```

- Scrittura di stringhe

```
int puts( char *s );
```

Header stdio.h

- Output Formattato

```
int printf( “stringa di controllo”, elencoVaribili );
```

- Formati importanti

```
%c          singolo carattere
%d, %i      intero con segno
%e, %E      notazione scientifica 1e+10
%f          virgola mobile 1.25
%s          stringa di caratteri
%u          intero senza segno
%x, %X      esadecimale xF5
```

- Esempio

```
printf(“singolo carattere:%c\n”,carat);
printf(“intero con segno:%d\n”,indice);
printf(“numero decimale: %f\n”,valore);
```

Header stdio.h

- Input formattato

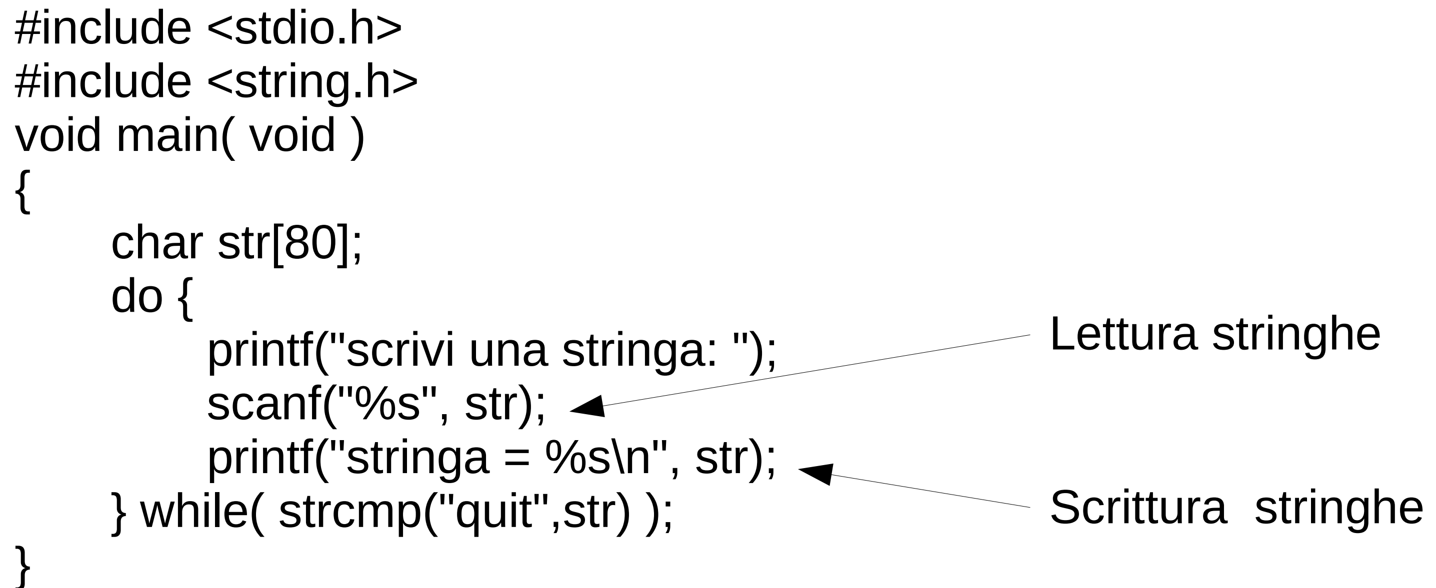
```
int scanf( char * stringaDiControllo, indirizzoVariabili)
```

- Le variabili di scanf sono dati mediante indirizzo
- I formati sono gli stessi di printf
- Esempio:

```
int x; char nome[10];
char carat; float nr;

scanf(“%d”, &x);
scanf(“%s”, nome);      <- Lettura stringa
scanf(“%c”, &carat);
scanf(“%f”, &nr);
```

Header stdio.h

- Lettura/scrittura stringhe:



```
#include <stdio.h>
#include <string.h>
void main( void )
{
	char str[80];
	do {
		printf("scrivi una stringa: ");
		scanf("%s", str);
		printf("stringa = %s\n", str);
	} while( strcmp("quit",str) );
}
```