# Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli


SAPIENZA
UNIVERSITÀ DI ROMA

Corso di C

Christian Cardia, Gabriele Saturni

# Introduzione al linguaggio C
## Perché impararlo?

- Il C è un linguaggio di programmazione sviluppato nel laboratorio AT&T della Bell nel 1972
- C largamente usato in molti ambiti
  - Telecomunicazioni;
  - Controllo di processi industriali;
  - embedded systems;
  - Sviluppo e testing di protocolli di rete
  - etc.
- Il C è un linguaggio minimalista e molto efficiente

# Introduzione al linguaggio C

## Il primo programma in C

Il primo esempio di programma in C sarà il classico “Hello World!”

# Introduzione al linguaggio C

## Il nostro primo programma in C

```c
//Questo programma stampa Hello World!
#include <stdio.h>
/*la funzione main () denota l'inizio dell'esecuzione
del programma.
VA SEMPRE INSERITO in ogni tipo di programma C*/
int main () {
    printf("Hello World! \n");
    return 0;
}
```

# Introduzione al linguaggio C

## Il nostro primo programma in C – Cosa abbiamo scritto ?

- Ogni ist … sintassi:
- Cosa s…
  - Sono … menti.
  - // com…
  - /* */ u…
- **I comm… sono fondamentali quando il codice diventa complesso!**

# Introduzione al linguaggio C

## Direttive al preprocessore

- #include <stdio.h>
- Il preprocessore viene invocato prima della compilazione ed effettua delle manipolazioni sul codice che devono essere effettuate prima della compilazione
  - Esempio: includere parti nel file prima della compilazione –header di librerie o effettuare delle sostituzioni di testo, come togliere i commenti).
- In particolare, la direttiva stdio.h permette l'inclusione delle funzioni di input/output (printf, scanf, etc.)
- Concetto simile alle *import* nel linguaggio JAVA

```
1  //Questo programma stampa Hello World!
2  #include <stdio.h>
3  /*la funzione main () denota l'inizio dell'esecuzione
4  del programma.
5  VA SEMPRE INSERITO in ogni tipo di programma C*/
6  int main () {
7    printf("Hello World! \n");
8    return 0;
9  }
```

# Introduzione al linguaggio C
## Il main

- Il **main** fa parte di ogni programma C. Le parentesi graffe che seguono indicano un blocco di istruzioni da eseguire.
- Un programma può contenere più funzioni ciascuna delle quali svolge un compito.
- Il main è sempre presente ed è la prima funzione eseguita. Altre funzioni possono essere invocate all'interno del main per svolgere dei sotto-compiti.

# Introduzione al linguaggio C

## Stampare: printf

- Funzione che stampa una stringa (sequenza di caratteri) indicata tra i doppi apici.
- *\n* non viene stampato in output ma permette di andare a capo
- A breve vedremo questa funzione più nel dettaglio

```c
//Questo programma stampa Hello World!
#include <stdio.h>
/*la funzione main () denota l'inizio dell'esecuzione
del programma.
VA SEMPRE INSERITO in ogni tipo di programma C*/
int main () {
  printf("Hello World! \n");
  return 0;
}
```

# Introduzione al linguaggio C

## Return 0;

- La parola chiave **return** indica il valore restituito dalla funzione
- In questo caso main() chiede che venga restituito il valore intero 0
- Nel main 0 è il valore di default. Indica che il programma è terminato con successo

```c
//Questo programma stampa Hello World!
#include <stdio.h>
/*la funzione main () denota l'inizio dell'esecuzione
del programma.
VA SEMPRE INSERITO in ogni tipo di programma C*/
int main () {
  printf("Hello World! \n");
  return 0;
}
```

# Introduzione al linguaggio C

## Eseguiamo il nostro primo programma

- Ora siamo pronti per eseguire il nostro primo programma in C !

# Introduzione al linguaggio C

## Il nostro primo programma in C

- Scrivete il programma (con un editor).
- Salvatelo in una cartella e chiamate il file primoprogramma.c
- Aprite il terminale, andate nella cartella (usando il comando cd <percorso della cartella>, e digitate:

  1) gcc -o primoprogramma  primoprogramma.c

  Il comando genera un file eseguibile chiamato ./primoprogramma

  2) Digitate ./primoprogramma e vedrete stampata la stringa *Hello World!*

# Introduzione al linguaggio C
## La compilazione

- La compilazione è un processo informatico che traduce una serie di istruzioni scritte in un determinato linguaggio di programmazione (codice sorgente) in istruzioni di un altro linguaggio (codice oggetto)!

- Il codice oggetto successivamente potrà essere eseguito dal computer.

# Introduzione al linguaggio C

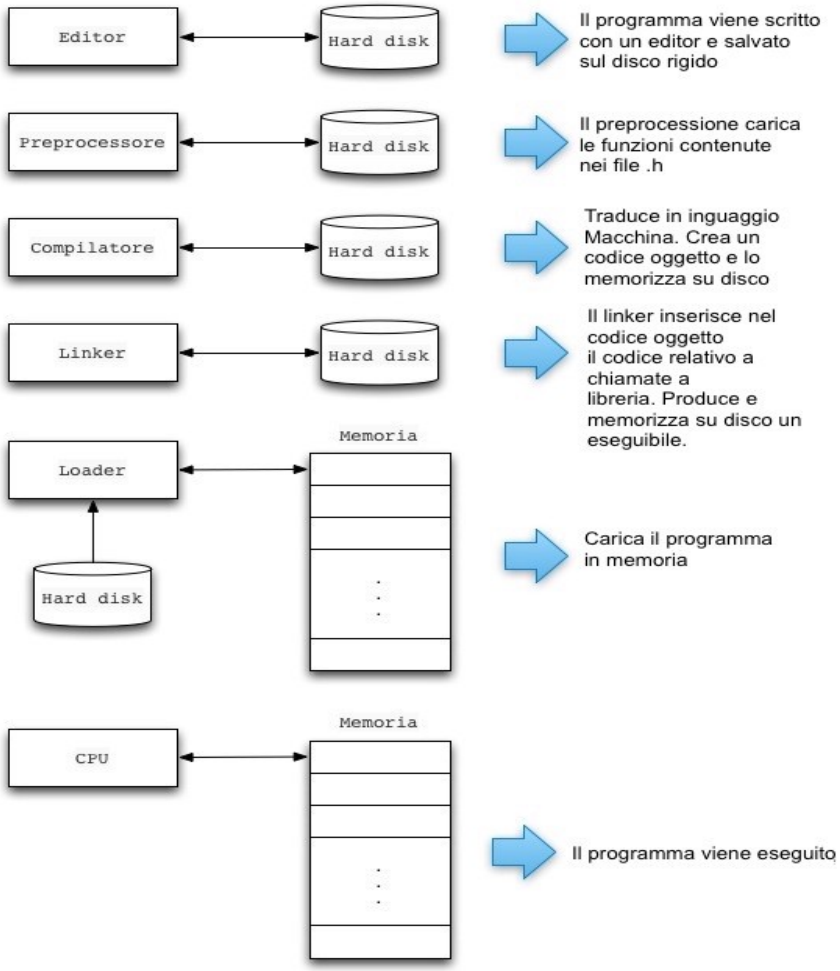
## La compilazione

Un programma C per essere eseguito segue le seguenti fasi:

1. Scrittura
2. Preprocessing
3. Compilazione
4. Link
5. Load
6. Esecuzione

# Introduzione al linguaggio C

## Le fasi della compilazione



# Introduzione al linguaggio C
## Dichiarazione di variabili

- Prima di poter utilizzare una variabile, la si deve dichiarare indicandone nome e tipo. La sintassi per dichiarare una variabile è:
  - <tipo variabile> <nome>;
- Ad esempio:
  - int x; → dichiara una variabile di nome x di tipo intero
- E' una buona abitudine inizializzare SEMPRE le variabili al momento della loro dichiarazione.
- Ad esempio:
  - int x=0; → assegna il valore 0 alla variabile x

# Introduzione al linguaggio C
## Assegnamento

- Il contenuto di una variabile si cambia usando un ***assegnamento*** con la sintassi:
  - <tipo variabile> = <valore>;
- Ad esempio:
  - x=5; → memorizza il valore 5 nella locazione di memoria indicata da x. Il valore della variabile cambia subito dopo l'assegnamento e rimane tale fino all'assegnamento successivo.

# Introduzione al linguaggio C

## Operatori aritmetici

- Per fare calcoli, si può operare su variabili e costanti usando gli stessi operatori aritmetici che si usano in matematica. Espressioni numeriche si possono scrivere in C con la seguente sintassi:
  - <espressione> <operatore_aritmetico> <espressione>
- In particolare il C fornisce i seguenti operatori aritmetici:
  - Somma → x+10
  - Differenza → x-10
  - Prodotto → x*10
  - Divisione intera → x/10
  - Modulo → x%10 → fa la divisione e restituisce il resto
  - Parentesi → 10*(x+2)
  - Incremento → x++ equivalente a x=x+1
  - Decremento → x-- equivalente a x=x-1

# Introduzione al linguaggio C
## Indirizzo di variabili

- In C possiamo accedere non solo al *valore* di una variabile ma anche al suo *indirizzo* → locazione in memoria a cui la variabile si trova. Per ottenere l'indirizzo di una variabile usiamo l'operatore &:
  - &<variabile>

- Ad esempio &x si riferisce all'indirizzo della variabile x. Vedremo più avanti come usare questi indirizzi in generale, che per ora utilizzeremo per input e output.

# Input & Output

## Stampare a video: printf

- Abbiamo già visto che la printf si usa per stampare a video un messaggio…ma può essere usata anche per stampare i valori di una o più variabili.

- La funzione printf ha la seguente sintassi:

  printf(<argomento_1>)

  printf(<argomento_1>,<argomento_2>)
  printf(<argomento_1>,…,<argomento_n>)

# Input & Output

## Stampare a video: printf

- **printf** ha uno o più argomenti. Il primo argomento è una sequenza di caratteri, delimitati da ", che indica cosa si vuole stampare. Gli altri argomenti sono variabili o espressioni di cui si vuole stampare il valore.

- Quindi:
  - per stampare un valore di una variabile di tipo int, si usa il simbolo %d nella stringa di caratteri.
  - printf("Stampiamo in valore di x: %d", x);

# Input & Output

## Inserire un input da tastiera: scanf

- In C per inserire un input da tastiera bisogna usare la funzione **scanf**, contenuta in stdio.h

- Il formato della funzione scanf è simile alla printf:
  - Il primo argomento è una sequenza di caratteri di escape che indica cosa si vuole inserire in input. Gli altri argomenti sono variabili nelle quali memorizzare il valore inserito.

- Per far ciò, scanf si deve riferire all'indirizzo di queste variabili. Il formato è il seguente:
  - scanf(<stringa>,<indirizzi_di_variabili>);

Quindi, per leggere un valore int e salvarlo nella variabile x si usa il comando:

- scanf('%d', &x);

# Esecuzione sequenziale

- Le istruzioni C che abbiamo coperto finora (dichiarazioni e assegnamenti), vengono eseguiti in modo sequenziale, uno dopo l'altro. Ad esempio, il codice:

```
int x, y;
x = 10;
y = x * 5;
x = y + 10;
```

- inizia assegnando a x il valore 10, per poi assegnare a y il valore 50 (10, memorizzato in x, per 5). Il valore di x cambia poi nell'ultima riga a 60 (50, memorizzato in y, più 10).

# Introduzione al linguaggio C

## Tipi primitivi

- Il C definisce vari *tipi primitivi*, che chiamiamo così perché sono definiti nel linguaggio stesso.
- In C (e in tutti gli altri linguaggi di programmazione), i numeri non hanno precisione infinita.
- Ogni variabile occupa un quantità fissa di memoria specificata dal tipo di dati.
  - Ad esempio una variabile di tipo **int** occupa 4 bytes, equivalenti a 32 bits e può rappresentare solo $2^{32}$valori diversi.

# Introduzione al linguaggio C
## Tipi primitivi (2)

- Il C offre vari tipi per rappresentare numeri interi. In genere, useremo il tipo int

| tipo | bits | bytes | min | max | printf/scanf |
|---|---|---|---|---|---|
| `char` | 8 | 1 | -127 | 127 | `"%hhd"` |
| `short` | 16 | 2 | -32.767 | 32.767 | `"%hd"` |
| `int` | 32 | 4 | -2.147.483.647 | 2.147.483.647 | `"%d"` |
| `long` | 32 | 4 | -2.147.483.647 | 2.147.483.647 | `"%ld"` |
| `long` | 64 | 8 | $-2^{63}$ | $2^{63}$ | `"%lld"` |
| `long` | | | | | |

# Introduzione al linguaggio C
## Tipi primitivi (2)

- Se non abbiamo la necessità di rappresentare numeri negativi, possiamo dedicare tutti i bit del tipo a numeri positivi, per un modesto incremento di precisione.

| tipo | bits | bytes | min | max | printf/scanf |
|---|---|---|---|---|---|
| `unsigned char` | 8 | 1 | 0 | 255 | `"%hhu"` |
| `unsigned short` | 16 | 2 | 0 | 65.535 | `"%hu"` |
| `unsigned int` | 32 | 4 | 0 | 4.294.967.295 | `"%u"` |
| `unsigned long` | 32 | 4 | 0 | 4.294.967.295 | `"%lu"` |
| `unsigned long long` | 64 | 8 | 0 | $2^{64}-1$ | `"%llu"` |

# Introduzione al linguaggio C
## Tipi primitivi (3)

- Per rappresentare numeri frazionari, il calcolatore usa la rappresentazione in virgola mobile (*floating point*).
- Parte dei bits sono dedicati a rappresentare il numero frazionale, la *mantissa*, e un'altra parte sono usati per rappresentare un *esponente*. I numeri in virgola mobile sono espressi come $m \cdot b^e$ dove $m$ è la mantissa, $b$ la base e $e$ l'esponente.

| tipo | bits | bytes | prec. | min e. | max e. | printf/scanf |
|---|---|---|---|---|---|---|
| `float` | 32 | 4 | 6 | -37 | 38 | `"%f"` |
| `double` | 64 | 8 | 15 | -307 | 308 | `"%lf"` |
| `long double` | 128 | 16 | 18 | -4931 | 4932 | `"%Lf"` |

# Introduzione al linguaggio C
## Promozione di Tipi

- Quando una operazione ha operandi di tipo diverso, il C converte automaticamente il tipo più piccolo a quello più grande. Formalmente diciamo che avviene una *promozione* da un tipo all'altro.

- Ad esempio, se si somma una variabile **short** and una **int**, la variabile short viene convertita ad **int** automaticamente dal linguaggio.

# Introduzione al linguaggio C
## Il cast

- A volte può interessare effettuare dei cambiamenti da un tipo *meno capiente* ad un tipo *più capiente*. In tali casi il linguaggio C mette a disposizione del programmatore un costrutto chiamato ***cast***.

```
1   int uno, due;
2   float tre;
3
4   uno = 1;
5   due = 2;
6   tre = (float) uno/due;
7   printf("%f",tre);
```

# Esercizio
## Hands on C

Scrivere un programma C che prende in input due numeri interi da tastiera e:

1. Calcola la somma tra i due numeri e stampa il risultato
2. Calcola il prodotto tra due numeri e stampa il risultato

# Esercizio

## Hands on C

```c
#include <stdio.h>

int main() {
    int n1, n2, somma, prodotto;
    printf("\nInserisci il primo numero: ");
    scanf("%d",&n1);
    printf("\nInserisci il secondo numero: ");
    scanf("%d", &n2);
    somma=n1+n2;
    prodotto=n1*n2;
    printf("\nLa somma dei due numeri e': %d", somma);
    printf("\nIl prodotto dei due numeri e': %d ", prodotto);
    printf("\n");
    return 0;
}
```

# Istruzione di selezione

Fino ad ora, abbiamo visto come eseguire istruzioni in modo sequenziale. Per fare scelte, dobbiamo introdurre l'istruzione di selezione.

**if** (<condizione>) {

   <istruzioni_se_condizione_vera>

} **else** {

   <istruzioni_se_condizione_falsa>

}

L'istruzione if valuta la condizione. Se questa risulta *vera*, le istruzioni dopo l'if vengono eseguite.

Se la condizione è *falsa*, le istruzioni dopo l'else vengono eseguite. La parte else puo' essere omessa.

# Istruzione di selezione (2)

Esiste anche una versione più succinta che viene usata quando c'è solo un'istruzione da eseguire nei blocchi.

**if** (<condizione>)

   <istruzione_se_condizione_vera>;

**else**

   <istruzione_se_condizione_falsa>;

*Sconsigliamo vivamente di usare questa forma in programmi complessi!* Potrebbe causare confusione e problemi inaspettati nell'esecuzione dei programmi (ad esempio se si aggiungono istruzioni dopo l'if). La mostriamo solo per completezza.

# Istruzione di selezione (3)

Un esempio:

- Dato un numero n in input controllare se è pari o dispari.

```c
#include <stdio.h>

//confronta se un numero è pari o dispari
int main() {
    int n;
    printf("Inserisci un numero ");
    scanf("%d", &n);
    if(n % 2 == 0 ) {
        printf("\nIl numero %d e' pari \n",n);
    }
    else {
        printf("\nIl numero %d e' dispari \n",n);
    }
    return 0;
}
```

# Operatori relazionali

- Per esprimere condizioni tra espressioni si usano gli operatori relazionali con la seguente sintassi:

  <*espressione*> <*operatore_relazionale*> <*espressione*>

- In particolare, il linguaggio C definisce i seguenti operatori relazionali:
  - minore: 5 < x, che risulta *vero* per x uguale a 10
  - maggiore: 5 > x, che risulta *falso* per x uguale a 10
  - minore o uguale: 5 <= x, che risulta *vero* per x uguale a 5
  - maggiore o uguale: 5 >= x, che risulta *vero* per x uguale a 5
  - uguale: 5 == x, che risulta *vero* per x uguale a 5
  - diverso: 5 != x, che risulta *falso* per x uguale a 5

# Esercizio
## Hands on C

- **Esercizio 1**: Scrivere un programma che legge tre numeri e li stampa in ordine crescente.

- **Esercizio 2**: Scrivere un programma che legge due date nel formato gg/mm/aaaa (ad es. 17/4/2009), le confronta e stampa se la prima data è anteriore, posteriore o uguale alla seconda. Ad esempio, se le date sono 12/5/2009 e 10/7/2009, il programma deve stampare la prima data è anteriore alla seconda.
  - **N.B: potete usare tranquillamente i / nell'input della scanf.**

# Operatori logici

- Varie condizioni di possono combinare usando gli operatori logici.
- Il C definisce gli operatori logici:
  - *AND* → &&
  - *OR* → ||
  - *NOT* → !

| a | b | *and*: a && b | *or*: a \|\| b | *not*: ! a |
|---|---|---|---|---|
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

# Istruzioni Iterative

## Ciclo for

- In quasi tutti i programmi avremo la necessità di ripetere un'istruzione un numero variabile di volte. Per far ciò, si utilizzano le *istruzioni iterative*. La prima istruzione di questo tipo e' il ciclo **for**.

**for**(<inizializzazione>; <condizione>; <espressione iterativa>) {
<istruzioni>
}

- Nel ciclo for, le istruzioni sono ripetute continuamente finchè condizione è vera. In genere, si controlla l'esecuzione del ciclo usando variabili addizionali, dette *contatori*, che vengono inizializzate e incrementate o decrementate.

# Istruzioni Iterative
## Ciclo for

Un esempio:

- Scrivere un programma che prende in input n numeri in virgola mobile (float) e stampa la media di tali numeri.

# Istruzioni Iterative

## Ciclo for

Un esempio:

•Scrivere un programma che prende in input n numeri in virgola mobile (float) e stampa la media di tali numeri.

```c
#include <stdio.h>

//stampa la media di n numeri
int main() {
    int n;
    printf("Quanti numeri? ");
    scanf("%d", &n);
    float somma = 0;
    for (int i = 1 ; i <= n ; i++) {
        printf(" %d: ", i);
        float x;
        scanf("%f", &x);
        somma += x;
    }
    printf("La media è %f\n", somma/n);
    return 0;
}
```

# Istruzioni Iterative

## Ciclo while

- L’altra istruzione iterativa messa a disposizione dal C è il ciclo **while.**
- Il while viene usato per ripetere un blocco di istruzioni finche' una condizione è vera.

```
while(<condizione>) {
    <istruzioni>
}
```

# Istruzioni Iterative
## Ciclo while

Un esempio:

- Programma che prende in input un intero n e stampa il numero di cifre di n.

# Istruzioni Iterative

## Ciclo while

Un esempio:

- Programma che prende in input un intero n e stampa il numero di cifre di n.

```c
#include <stdio.h>

// stampa il numero di cifre di un intero non negativo
int main() {
    long long n;
    printf("Digita un intero non negativo: ");
    scanf("%lld", &n);
    int nc = 1;
    while (n > 9) {
        nc++;
        n /= 10;
    }
    printf("Il numero di cifre è %d\n", nc);
    return 0;
}
```

# Istruzioni Iterative
## Ciclo do-while

- Il ciclo **do - while** ha la stessa semantica del ciclo **while** solo che il blocco di istruzioni viene eseguito almeno una volta.

```
do {
    <istruzioni>
    } while (<condizione>);
```

# Istruzioni Iterative
## Ciclo do-while

Un esempio:

- Programma che legge una linea di testo (sequenza di caratteri terminata dal carattere '\n') e conta il numero di vocali e di consonanti.

# Istruzioni Iterative

## Ciclo do-while

Un esempio:

- Programma che legge una linea di testo (sequenza di caratteri terminata dal carattere '\n') e conta il numero di vocali e di consonanti.

```c
#include <stdio.h>

//conta il numero di vocali e consonanti in una linea di testo
int main() {
    printf("Inserire una linea di testo: ");
    char c;
    int vocali = 0, cons = 0;
    do {
        scanf("%c", &c);
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
            vocali++;
        else if (c > 'a' && c <= 'z') cons++;
    } while (c != '\n');
    printf("vocali: %d   consonanti: %d\n", vocali, cons);
    return 0;
}
```

# Istruzioni Iterative

## Equivalenza tra le istruzioni iterative

- I tre cicli introdotti sono equivalenti.
- Esistono linguaggi di programmazione che non li hanno tutti.
- Il C ne introduce tre per concedere al programmatore di usare il ciclo sintatticamente più corto. In inglese si direbbe "syntactic sugar".

# Istruzioni Iterative

## Equivalenza tra le istruzioni iterative

```
while (C) { B }
```

```
for (; C; ) { B }
```

```
for (I; C; E) { B }
```

```
I; while (C) { B E }
```

```
do { B } while(C);
```

```
B; while (C) { B }
```

```
while (C) { B }
```

```
if (C) { do { B } while (C); }
```

# Gli Array

## Definizione

- Un Array è una collezione di dati omogenei
- Vengono memorizzati in celle successive di memoria
- È possibile accedere in tempo costante ad ogni elemento, specificando la sua posizione
- Esempi di Array:

| 3 | 23 | 1 | 14 | 22 |
|---|---|---|---|---|

Array di interi

Array di char

| A | E | s | Q | P |
|---|---|---|---|---|

# Gli Array

## Dichiarazione di un Array

Dichiarazione di un Array:

- *tipo nome [dimensione];*

```
 6      //1. array di int con 10 posizioni
 7      int i [10];
 8      //2. array di char di 5 posizioni
 9      char c [5];
10      //3. array di int inizializzato
11      int numeri [ ] = {7,2,3,6};
12      //4. array di char inizializzato
13      char caratteri [ ] = {'h','e','l','l','o'};
14      //5. array di char inizializzato
15      char caratteri2 [ ] = "Hello";
```

# Gli Array

## Accedere agli elementi di un Array

- È possibile accedere ad ogni elemento dell'Array in tempo costante indicando la sua posizione
- Un Array con *n* elementi ha gli elementi che vanno dalla posizione *0* alla posizione *n-1*
- L'indice dell'elemento dell'Array può anche essere specificato da un'espressione: a[2+3], a[4*3] ecc..

Es. dichiariamo l'Array *c* di tipo *int* e con 10 elementi

# Gli Array

## Accedere agli elementi di un Array

```
 5    //dichiarazione array di int di 10 elementi
 6    int array[10];
 7    //imposto il primo elemento a 3
 8    array[0] = 3;
 9    //imposto il secondo elemento a -3
10    array[1] = -3;
11    //imposto il settimo elemento a 5
12    array[6] = 5;
13    //dichiaraz variabile int e gli assegno il primo elemento
14    int a = array[0];
15    //dichiaraz variabile int e gli assegno il valore 2
16    int b = 2;
17    //equivale a scrivere 'array[5] = 3'
18    array[a+b] = 3;
19    //equivale a scrivere 'array[6]=(array[6] + 2)
20    array[a*b] += 2;
21    //stampo i valori
22    printf("array[5]=%d, array[6]=%d \n",array[5],array[6]);
```

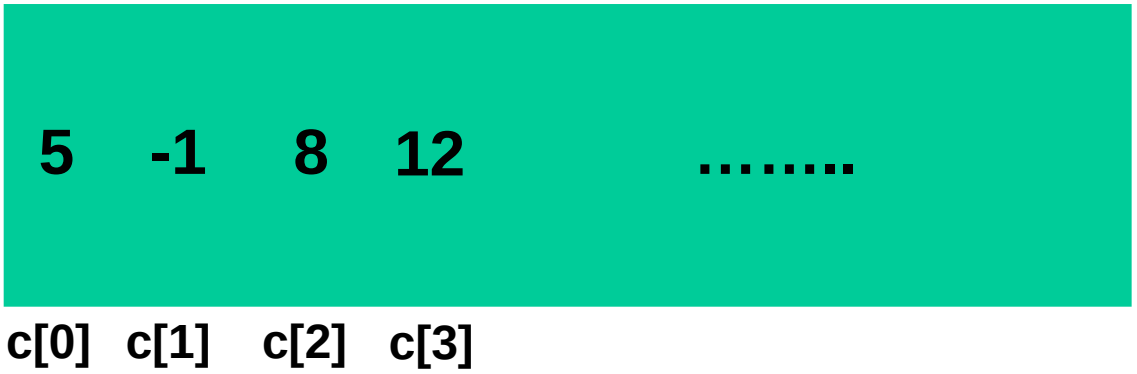
- **Output**: *array[5] =3, array[6]=7*

# Gli Array

## Accedere agli elementi di un Array

- Cosa succede se scriviamo queste righe di codice?

```
...
int a[4];
...inizializzo gli elementi...
printf("%d",a[4]);
...
```



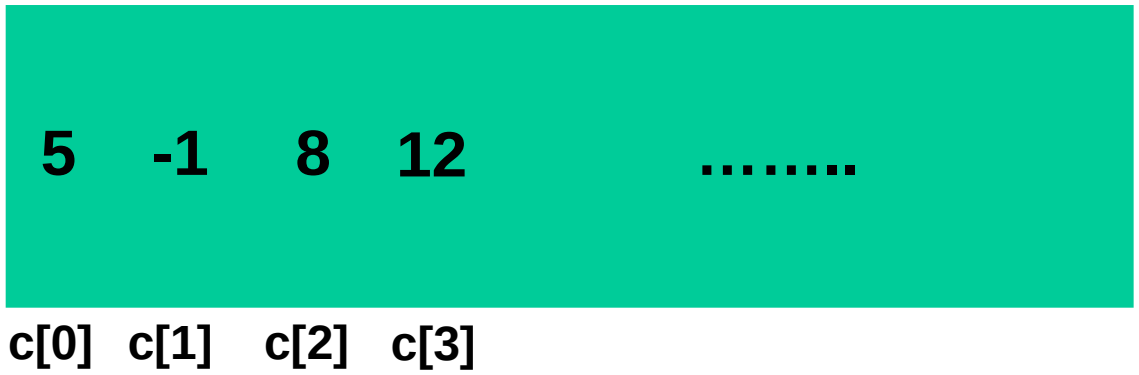
# Gli Array

## Accedere agli elementi di un Array

- Cosa succede se scriviamo queste righe di codice?

```
...
int a[4];
...inizializzo gli elementi...
printf("%d",a[4]);
...
```

Nessun errore in compilazione, ma il risultato è imprevisto!



# Gli Array

## Scorrere gli elementi di un Array tramite un ciclo for

```
 5      int array[10];
 6      int i = 0;
 7
 8      for(i=0;i<10;i++){
 9          printf("Inserire elemento %d: ",i);
10          scanf("%d",&array[i]);
11      }
12
13      for(i=0;i<10;i++){
14          printf("array[%d]: %d \n",i,array[i]);
15      }
16
```

## *Gli Array multidimensionali*

- È possibile rappresentare tabelle e matrici
- Si possono rappresentare strutture che hanno più di una dimensione
- Esempio: dichiarare una matrice composta da *N* righe e *M* colonne:

```
int matrix[n][m]; //n=numero righe, m=numero colonne
```

- Per accedere agli elementi è opportuno specificare l'indice della riga e della colonna:

```
matrix[0][1] = 4;
printf("%d",matrix[0][1]);
```

**Elemento alla riga 0 e alla colonna 1**

# *Gli Array multidimensionali*

*Esempio*

```
 5      int matrice[2][3];
 6
 7      int riga = 0;
 8      int colonna = 0;
 9
10      for ( riga=0; riga<2; riga++ ) {
11          for ( colonna=0; colonna<3; colonna++ ) {
12              printf("Input [%d][%d] \n",riga,colonna);
13              scanf("%d",&matrice[riga][colonna]);
14          }
15      }
16
17      for ( riga=0; riga<2; riga++ ) {
18          for ( colonna=0; colonna<3; colonna++ ) {
19              printf("%d ",matrice[riga][colonna]);
20          }
21          printf("\n");
22      }
```

# Esercitazione

**Esercizio**. Scrivere un programma che legge un carattere c e un intero n e stampa un triangolo di altezza n fatto con caratteri c.

# Esercitazione

**Esercizio**. Scrivere un programma che prende in input un intero n e stampa la somma delle cifre di n. Ad esempio, se n = 1205 allora il programma stampa 8.

# Esercitazione

**Esercizio**. Scrivere un programma che legge una linea di testo e stampa la lunghezza della più lunga parola contenuta nella linea di testo. Ad esempio, se la linea di testo è: «Qual è la parola più lunga?» allora il programma stampa 6 se invece la linea di testo è «Una parola lunghissima», stampa 11.

# Esercitazione

**Esercizio.** Scrivere un programma che legge una linea di testo e stampa la somma dei numeri contenuti nella linea. Ad esempio se la linea è "L'appuntamento è alle 18:40 del 2/11/2010", allora il programma stampa 2081 (2081 = 18+40+2+11+2010).

## Esercitazione

- Si scriva un programma che dichiara un Array di interi di 100 elementi
- Il programma, tramite un ciclo *for*, deve inserire ad ogni posizione il valore della posizione stessa. Es.
- *A[0]=0, A[1]=1,……A[99]=99*
- Successivamente deve stampare sullo schermo, tramite un secondo ciclo *for*, tutti gli elementi dell'Array in ordine inverso (dall'ultimo elemento al primo)

# Esercitazione
## Soluzione

```
 6      int array[100];
 7      int i;
 8
 9      for(i=0;i<100;i++){
10          array[i] = i;
11      }
12
13      for(i=99;i>-1;i--){
14          printf("array[%d]= %d \n",i,array[i]);
15      }
16
```

## Esercitazione

- Si scriva un programma che dichiara una matrice di interi con 3 righe e 4 colonne:
- Il programma deve permettere all'utente di inserire in input gli elementi della matrice
- Successivamente deve stampare sullo schermo tutti gli elementi della matrice **pari**
- Per fare l'esercizio utilizzare il ciclo *for*

# Esercitazione

## Soluzione

```
 5      int matrice[3][4];
 6
 7      int riga = 0;
 8      int colonna = 0;
 9
10      for (riga=0; riga<3; riga++){
11          for(colonna=0;colonna<4;colonna++){
12              printf("Input[%d][%d] \n",riga,colonna);
13              scanf("%d",&matrice[riga][colonna]);
14          }
15      }
16
17      printf("Numeri pari: \n");
18
19      for (riga=0; riga<3; riga++){
20          for(colonna=0;colonna<4;colonna++){
21              int numero = matrice[riga][colonna];
22              if( (numero%2)==0 ){
23                 printf("%d \n",matrice[riga][colonna]);
24              }
25          }
26      }
```