



Introduzione al C

InformaticaB

Facciamo un passo indietro...

Compilare o Interpretare ?

- ❑ Come rendo “comprensibili” al calcolatore i programmi scritti in un linguaggio di programmazione (ad es. C, C++, Java) ?
- ❑ Due sono gli approcci maggiormente usati:
 - ▶ Utilizzando un **compilatore**, un programma che traduce i programmi di alto livello in codice macchina
 - ▶ Utilizzando un **interprete**, un programma che interpreta direttamente le istruzioni di alto livello e le esegue
- ❑ Entrambi gli approcci offrono vantaggi e svantaggi
 - ▶ Portabilità
 - ▶ Efficienza
 - ▶ Controlli
 - ▶ Semplicità

II C?

La catena di sviluppo in C

- ❑ Il C è un linguaggio compilato
- ❑ Possiamo individuare 5 passi per passare dalla definizione di un algoritmo ad un programma in esecuzione che lo implementa



1. Scrittura

- ❑ Il programma, costituito da una sequenza di caratteri, viene **composto e modificato** usando un qualsiasi editor
- ❑ Così otteniamo un **codice sorgente** memorizzato in memoria di massa in un file di testo (es. XYZ.c)

2. Traduzione

- ❑ Il compilatore si occupa della traduzione dal linguaggio di alto livello al linguaggio macchina
- ❑ Durante questa fase si riconoscono i simboli, le parole e i costrutti del linguaggio:
 - ▶ eventuali messaggi diagnostici segnalano errori di sintassi
- ❑ Viene generato il codice macchina in forma binaria : a partire dal **codice sorgente** si genera il **codice oggetto**, cioè in un file **binario**

3. Collegamento (linking)

- ❑ Il collegatore (linker) deve collegare fra loro il file oggetto ed altre librerie utilizzate (es. librerie di I/O)
- ❑ Si rendono globalmente coerenti i riferimenti agli indirizzi dei vari elementi collegati
- ❑ Si genera un **programma eseguibile**, un file binario che contiene il codice macchina del programma eseguibile completo, di nome XYZ.exe
- ❑ Messaggi di errore possono essere dovuti ad errori nel citare i nomi delle funzionalità di librerie esterne da collegare
- ❑ Il programma sarà effettivamente eseguibile solo dopo che il contenuto del file sarà stato caricato nella memoria di lavoro (centrale) del calcolatore

4. Caricamento (loading)

- ❑ Il caricatore (*loader*) individua una porzione libera della memoria di lavoro e vi copia il contenuto del programma eseguibile
 - ▶ Eventuali messaggi rivolti all'utente possono segnalare che non c'è abbastanza spazio in memoria

5. Esecuzione

- ❑ Per eseguire il programma occorre fornire in ingresso i dati richiesti e in uscita riceveremo i risultati (su video o file o stampante)
- ❑ Durante l'esecuzione possono verificarsi degli errori (detti "errori di run-time"), quali:
 - ▶ calcoli con risultati scorretti (per esempio un overflow)
 - ▶ calcoli impossibili (divisioni per zero, logaritmo di un numero negativo, radice quadrata di un numero negativo,....)
 - ▶ errori nella concezione dell'algoritmo (l'algoritmo non risolve il problema dato)
- ❑ Tutti gli esempi citati si riferiscono ai cosiddetti *errori semantici*

Scriviamo il nostro primo programma C

Struttura di un programma C (vs Matlab)

- ❑ In C non esiste un interprete interattivo: è necessario scrivere un codice sorgente e compilarlo
- ❑ Il codice sorgente di un programma è organizzato in una o più funzioni
 - ▶ non esistono gli script
 - ▶ ogni programma deve contenere la funzione **main**
 - ▶ quando un programma viene eseguito, vengono eseguite le istruzioni contenute nella funzione **main**

Il nostro primo programma in C

```
/* Questo è il nostro primo  
   programma in C */
```

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

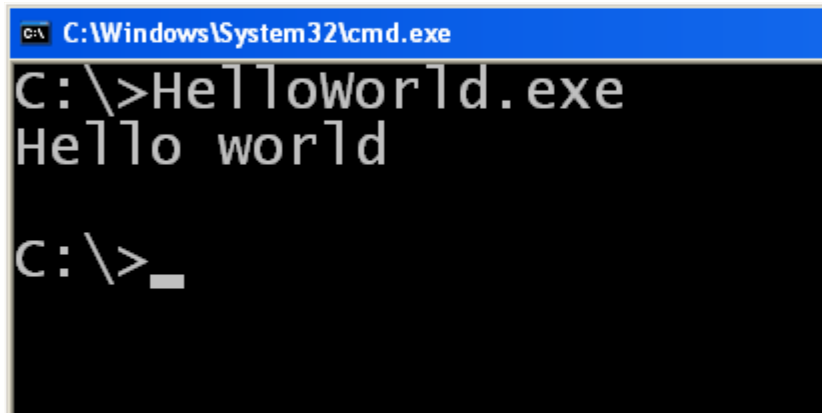
```
int main( )
```

```
{
```

```
    printf("Hello world\n");
```

```
    return 0;
```

```
}
```



```
C:\Windows\System32\cmd.exe  
C:\>HelloWorld.exe  
Hello world  
C:\>_
```

Il nostro primo programma in C

```
/* Questo è il nostro primo  
programma in C */
```

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

```
int main()  
{  
    printf("Hello world\n");  
    return 0;  
}
```

- ❑ Questo è un commento
- ❑ Tutto ciò che viene racchiuso fra */** e **/* viene ignorato dal compilatore:

```
/* ... commento ... */
```

- ❑ */** apre il commento
- ❑ **/* chiude il commento
- ❑ Viene utilizzato per aggiungere note e/o descrivere parti del programma

Il nostro primo programma in C

```
/* Questo è il nostro primo  
programma in C */
```

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

```
int main()  
{  
    printf("Hello world\n");  
    return 0;  
}
```

- ❑ Indica al compilatore che nel nostro programma verranno utilizzate delle **funzioni esterne**
- ❑ In questo caso specifica che queste funzioni si trovano nella **libreria `stdio.h`** (standard input output)
- ❑ Il C mette a disposizione del programmatore diverse librerie, cioè diverse collezioni di funzioni (I/O, operazioni matematiche, informazioni su data e ora, ecc.)
- ❑ Per usare una libreria nel nostro programma occorre specificarlo all'inizio:

```
#include<nome libreria>
```

Il nostro primo programma in C

```
/* Questo è il nostro primo  
programma in C */
```

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

```
int main()
```

```
{
```

```
    printf("Hello world\n");
```

```
    return 0;
```

```
}
```

- ❑ Ogni programma in C deve contenere una funzione principale, chiamata **main**
- ❑ Questa funzione contiene le istruzioni che verranno eseguite non appena il nostro programma viene caricato in memoria
- ❑ La funzione **main** restituisce un valore intero

Il nostro primo programma in C

/ Questo è il nostro primo programma in C */*

`#include <stdio.h>`

`int main()`

```
{  
    printf("Hello world\n");  
    return 0;  
}
```

□ Sintassi funzioni C vs MATLAB

```
tipo nomef (<params>)  
{  
    /* Codice funzione*/  
    return <value>;  
}
```

C

```
function ris = nomef (<params>)  
% Codice funzione  
ris = <value>
```

MATLAB

Il nostro primo programma in C

```
/* Questo è il nostro primo  
programma in C */
```

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

```
int main()
```

```
{
```

```
    printf("Hello world\n");
```

```
    return 0;
```

```
}
```

- ❑ La sequenza di istruzioni della funzione main (detto anche **corpo**) deve essere racchiuso all'interno di **parentesi graffe**
- ❑ Ciascuna istruzione (**statement**) deve essere seguita da un **punto e virgola**
- ❑ Più in generale, in C una sequenza di istruzioni racchiuse da parentesi graffe costituisce una sorta di "macroistruzione" detta **blocco**:

```
{  
    istruzione1;  
    ...  
    istruzionen;  
}
```

Il nostro primo programma in C

```
/* Questo è il nostro primo  
programma in C */
```

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

```
int main()
```

```
{
```

```
printf("Hello world\n");
```

```
return 0;
```

```
}
```

- ❑ Questa istruzione stampa a video il messaggio "Hello world"
- ❑ Utilizza la funzione `printf` fornita dalla libreria `stdio.h`
- ❑ La sintassi della funzione è la seguente:

```
printf ( "messaggio" )
```

- ❑ Il messaggio da stampare deve essere racchiuso da ""
- ❑ `\n` è una sequenza speciale (newline) che può essere usato per mandare il cursore a capo

Il nostro primo programma in C

```
/* Questo è il nostro primo  
   programma in C */
```

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


```
int main()  
{  
    printf("Hello world\n");  
    return 0;  
}
```

- ❑ L'istruzione **return** serve a terminare la funzione **main** ritornando un valore che rappresenta l'esito dell'esecuzione
- ❑ Nel caso della funzione **main** viene ritornato convenzionalmente il valore **0**, per indicare al sistema operativo che il programma è stato eseguito con successo
- ❑ Valori diversi possono essere usati per segnalare eventuali errori avvenuti

Output di un programma

```
/* Programma Room1.c */  
#include <stdio.h>
```

```
int main()  
{  
    printf("La mia stanza e' lunga %d metri e larga %d metri\n", 3, 4);  
    return 0;  
}
```



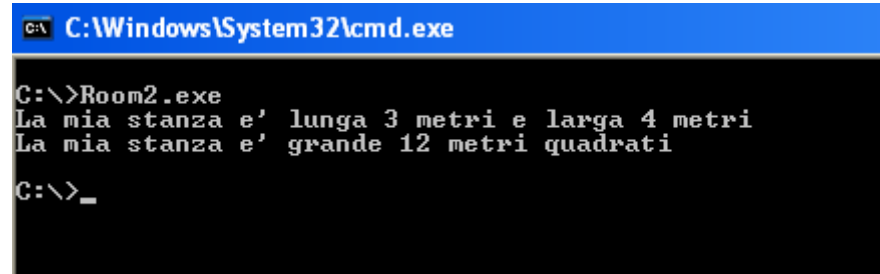
```
C:\>Room1.exe  
La mia stanza e' lunga 3 metri e larga 4 metri  
C:\>_
```

- ❑ La funzione printf consente di stampare anche dei valori numerici all'interno dei messaggi:
 printf("msg",valore1 ,valore2 ,...);
- ❑ Il primo argomento, detto **stringa di controllo**, può contenere **caratteri di formato** (preceduti da %) che verranno associati ai restanti argomenti della printf
- ❑ %d indica che l'argomento da stampare è un **numero intero**

Semplici calcoli

```
/* Programma Room2.c */  
#include <stdio.h>
```

```
int main()  
{  
    printf("La mia stanza e' lunga %d metri e larga %d metri\n",3,4);  
    printf("La mia stanza e' grande %d metri quadrati\n",3*4);  
    return 0;  
}
```



```
C:\Windows\System32\cmd.exe  
C:\>Room2.exe  
La mia stanza e' lunga 3 metri e larga 4 metri  
La mia stanza e' grande 12 metri quadrati  
C:\>_
```

- ❑ È anche possibile utilizzare come argomenti espressioni aritmetiche
- ❑ In questo caso, il calcolatore calcolerà l'espressione e poi utilizzerà il risultato come valore da stampare

Le variabili

- Affinché un programma sia in grado di risolvere una classe di problemi, occorre introdurre il concetto di **variabile**.

```
/* Programma Room3.c */
```

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

```
int main()
```

```
{
```

```
    int lung, larg, area;
```

```
    lung = 3;
```

```
    larg = 4;
```

```
    area = lung*larg;
```

```
    printf("La mia stanza e' lunga %d metri e larga %d metri\n",lung, larg);
```

```
    printf("La mia stanza e' grande %d metri quadrati\n",area);
```

```
    return 0;
```

```
}
```

Le variabili

```
/* Programma Room3.c */  
#include <stdio.h>
```

```
int main()  
{  
    int lung, larg, area;  
    lung = 3;  
    larg = 4;  
    area = lung*larg;  
    printf("La mia stanza e' lunga %d  
metri e larga %d metri\n",lung, larg);  
    printf("La mia stanza e' grande %d  
metri quadrati\n",area);  
    return 0;  
}
```

- ❑ Per usare una variabile in C, occorre dichiararla
- ❑ La dichiarazione di una variabile deve specificarne il **tipo** e il nome:

```
tipo nome1, nome2, ...;
```

- ❑ **int** indica al compilatore che la variabile conterrà un numero intero

Le variabili

```
/* Programma Room3.c */  
#include <stdio.h>
```

```
int main()  
{  
    int lung, larg, area;  
    lung = 3;  
    larg = 4;  
    area = lung*larg;  
    printf("La mia stanza e' lunga %d  
metri e larga %d metri\n",lung,larg);  
    printf("La mia stanza e' grande %d  
metri quadrati\n",area);  
    return 0;  
}
```

- In C è possibile assegnare un valore ad una variabile con la sintassi:

nome = <espressione>

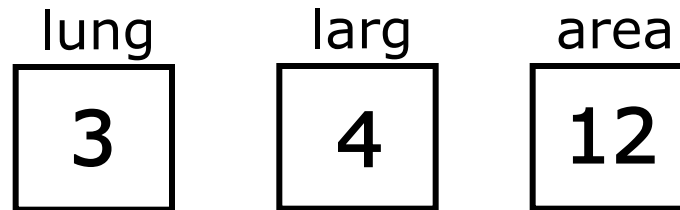
- ▶ Nome è il nome di una variabile
- ▶ Espressione è una qualsiasi espressione aritmetica valida in C (può contenere variabili)

- Il risultato dell'espressione viene messo nella variabile

nome ← <espressione>

Le variabili

```
int lung, larg, area;  
lung = 3;  
larg = 4;  
area = lung*larg;
```



Letture dati dal terminale

- ❑ Come posso fornire informazioni al mio programma in C?

```
/* Programma Room4.c */
```

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

```
int main()
```

```
{
```

```
    int lung, larg, area;
```

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

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

```
    area = lung*larg;
```

```
    printf("La mia stanza e' lunga %d metri e larga %d metri\n",lung,larg);
```

```
    printf("La mia stanza e' grande %d metri quadrati\n",area);
```

```
    return 0;
```

```
}
```

Letture dati dal terminale

```
/* Programma Room4.c */
```

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

```
int main()
```

```
{
```

```
    int lung, larg, area;
```

```
    scanf("%d",&lung);  
    scanf("%d",&larg);
```

```
    ...
```

- ❑ La lettura di informazioni dal terminale avviene tramite la funzione `scanf` fornita dalla libreria `stdio.h`
- ❑ La sintassi da utilizzare è:

 `scanf("str ctrl", &variabile)`
- ❑ Il primo parametro è la **stringa di controllo**
- ❑ È possibile definire un **pattern input** (es. leggere più variabili, leggere variabili in una specifica sequenza)
- ❑ In generale è preferibile utilizzare stringhe di controllo semplici e leggere una sola variabile per volta