Fasi per l'esecuzione di un programma

Un programma in C passa attraverso 6 fasi per essere eseguito:

1. stesura del programma tramite un editor di testi o un ambiente integrato di programmazione;
2. preelaborazione del programma da parte del preprocessore C;
3. compilazione del programma da parte del compilatore C;
4. collegamento del programma (linking) alle funzioni di libreria;
5. caricamento del programma eseguibile in memoria;
6. esecuzione del programma.

Un primo programma in C

```
/* Un primo programma in C */
#include <stdio.h>

/* La funzione main è il punto di inizio dell'esecuzione
   del programma */
int main( void )
{
   printf( "Welcome to C!\n" );

   return 0; /* il programma è terminato con successo */

} /* fine della funzione main */
```

La prima direttiva per il preprocessore

Il comando:

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

viene catturato dal preprocessore C (come tutte le righe di codice che iniziano con il carattere #)
Il preprocessore include all'interno del vostro codice il file di intestazione (stdio.h, dove l'estensione .h sta per "header") che contiene le segnature delle funzioni che utilizzerete per le operazioni elementari di input/output.

La funzione main

```
int main ( void ) { [corpo della funzione] }
```

La funzione main è il punto di partenza per l'esecuzione di un qualunque programma C.
La parola chiave `int` stabilisce il tipo di ritorno, nel caso della funzione main un intero.
Fra parentesi tonde, vengono elencati gli argomenti della funzione.
La parola chiave `void` indica che la funzione main non accetta alcun argomento in input (vedremo in seguito come passare degli argomenti alla funzione main).

Blocchi

Le parentesi graffe vengono usate per racchiudere un gruppo di istruzioni in un blocco. In questo caso, tutto ciò che si trova fra la parentesi graffa aperta e la chiusa, rappresenta il corpo delle istruzioni della funzione main.

La prima istruzione

```
pritnf("Welcome to C!\n");
```

La funzione `printf` è una delle funzioni di output fornite dalla libreria `stdio.h`.
Questa è la prima istruzione in C che vediamo. Ogni istruzione deve essere terminata da ";". Un'istruzione richiede al programma di eseguire un'azione; in questo caso di richiamare la funzione pritnf che stamperà su "stdout" la stringa passatale come argomento.

Compilare il primo programma

Una volta creato il file sorgente del nostro primo programma, dovremo salvarlo dandogli un nome e un'estensione `.c`.
Per compilare il programma usate il comando

```
gcc -o output.o input.c
```

L'opzione `-o` permette di definire il nome del file eseguibile in output (il default è `a.o`). `input.c` è il nome del file sorgente.
I comandi di shell di linux che possono servirvi per effettuare questa operazione sono:

- `pwd`: fornisce il percorso completo alla directory corrente;
- `ls`: elenca il contenuto della directory corrente;
- `cd nomecartella`: cambia directory (`..` porta alla directory superiore);
- `man nomecomando`: fornisce le informazioni sulla sintassi del comando.

Variabili

Una variabile è un riferimento ad una locazione in memoria nella quale è possibile immagazzinare dei valori, in maniera tale da renderne possibile l'utilizzo all'interno di un programma.
Una variable deve essere *dichiarata* prima del suo utilizzo, all'inizio del corpo della funzione.
Per dichiarare una variabile bisogna definirne il *tipo* ed assegnarle un *nome*.

```
int intero1;
int intero2;
int somma;
```

È possibile accorpare la dichiarazione di più variabili dello stesso tipo in una sola istruzione:

```
int intero1, intero2, somma;
```

Nomi delle variabili

Un nome di una varibile deve essere un *identificatore* valido:
un identificatore valido è una qualunque stringa di lettere, cifre e caratteri di sottolineatura ("_"). Il primo carattere non può essere una cifra.
Il C è case sensitive: `intero1`, `Intero1` e `iNtErO1` sono tre identificatori distinti, tutti e tre validi.
`1intero` non è un identificatore valido.

Input di valori interi

La funzione *scanf* permette di fornire dei valori in input ad un programma in esecuzione.
L'istruzione

```
scanf("%d", &intero1);
```

prende dati da *stdin* (la tastiera). La funzione ha 2 argomenti. Gli argomenti di una funzione sono separati da virgole.
Il primo argomento è una stringa: `"%d"` indica alla funzione scanf che il valore atteso in input è un intero.
Il secondo argomento è l'indirizzo di memoria in cui salvare il valore in input, ottenuto applicando l'operatore `&` alla variabile intera `intero1`.

Assegnamento di valori a variabili, operazioni aritmetiche elementari

Il C fornisce le operazioni aritmetiche elementari:

- somma: `+`
- sottrazione: `-`
- prodotto: `*`
- divisione: `/`
- resto: `%`

```
somma = intero1 + intero2;
```

L'operatore = per l'assegnamento di valori alle variabili associa da destra verso sinistra:

```
a=b=c=d=0;
```

assegna alle variabili `a`, `b`, `c` e `d` il valore 0.

Priorità degli operatori aritmetici

In un'espressione aritmetica complessa, le operazioni vengono eseguite in ordine di priorità, procedendo da sinistra verso destra.
La moltiplicazione, la divisione e il resto hanno priorità più alta della somma e della differenza.
Per modificare l'ordine di esecuzione delle operazioni è possibile utilizzare le parentesi tonde.

```
media = a + b + c / 3; <- sbagliato!

media = (a + b + c) / 3;
```

Operatori di confronto

Il C fornisce anche degli operatori per il confronto fra valori.
Operatori di uguaglianza:

- uguale: ==
- diverso: !=

Operatori relazionali:

- maggiore: >
- minore: <
- maggiore o uguale: >=
- minore o uguale: <=

Priorità degli operatori di confronto

Gli operatori di uguaglianza hanno la priorità più bassa di quelli di relazione che a loro volta hanno priorità più bassa di quelli aritmetici.
Associano tutti da sinistra a destra.

Il comando `if`

```
if (intero1 > intero2) { blocco istruzioni }
```

Il comando `if` permette di eseguire un blocco di istruzioni solamente nel caso la condizione, specificata fra parentesi tonde, sia soddisfatta (restituisca valore `true`).

Esercizio:

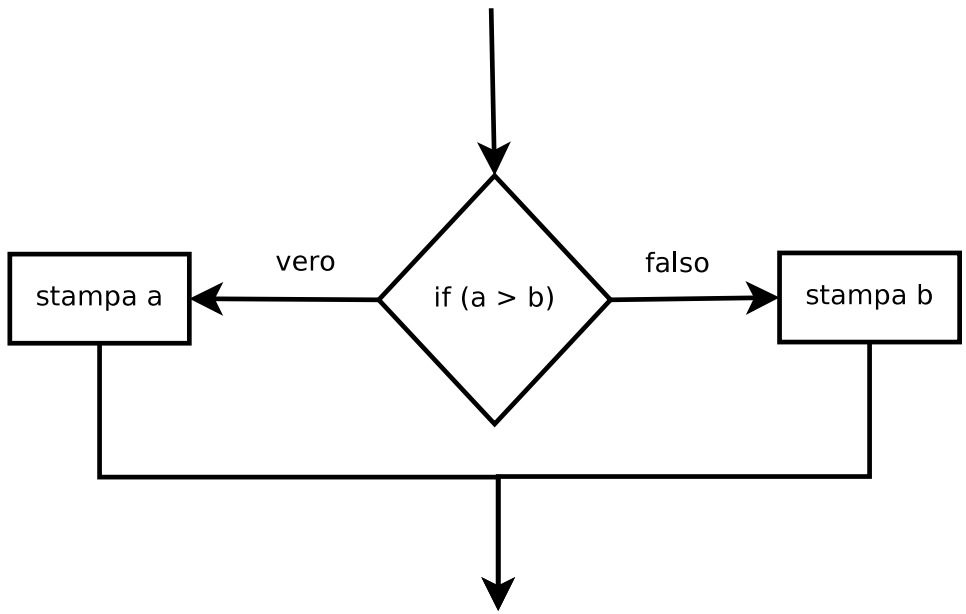
Scrivere un programma in C che richieda due valori interi maggiori o uguali a 0 in input e stampi in output il risultato della somma, della sottrazione e del prodotto fra questi due valori.
Il programma deve anche stampare il valore della divisione e del resto purchè il secondo valore sia maggiore di 0.

```
/* printf, scanf e comando if */
#include <stdio.h>
/* La funzione main è il punto di inizio dell'esecuzione
   del programma */
int main( void ) {
   int intero1; /* primo numero fornito dall'utente  */
   int intero2; /* secondo numero fornito dall'utente */

   printf( "Inserire due interi separati da spazio\n" );

   scanf( "%d%d", &intero1, &intero2 ); /* lettura di due numeri
                                           interi */
   printf("Il valore della somma e' %d\n", intero1 + intero2);
   printf("Il valore della differenza e' %d\n", intero1 - intero2);
   printf("Il valore del prodotto e' %d\n", intero1 * intero2);
   if ( 0 < intero2 ) { /* il corpo dell'if viene eseguito solo se
                           la condizione è true */
      printf("Il valore della divisione intera e' %d\n",
                                                 intero1 / intero2);
      printf("Il valore del resto e' %d\n", intero1 % intero2);
   } /* fine if */
   return 0; /* il programma è terminato correttamente */
} /* fine della funzione main */
```

Il comando `else`

```
if ( condizione ) { blocco istruizioni }
else { blocco istruzioni }
```

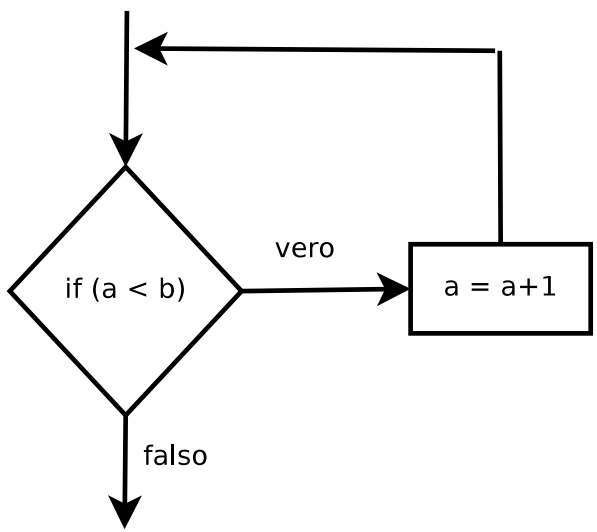
Il blocco di istruzioni dopo la parola chiave `else` viene eseguito solamente se la condizione del precedente comando `if` è falsa.



Il ciclo `while`

```
while ( condizione ) { blocco istruzioni }
```

Se la condizione del comando while è vera, viene eseguito il blocco di istruzioni. Al termine dell'esecuzione del blocco, la condizione viene verificata nuovamente e il ciclo si ripete finchè la condizione non diviene falsa.



Scorciatoie

`x [ + - * / % ]= 5`: scorciatoia per `x = x [ + - * / % ] 5`

`[ ++ -- ]x`: scorciatoia per `x = x+1 (x=x-1)`

`x[ ++ -- ]`: scorciatoia per `x = x+1(x=x-1)`

```
x = 5;
printf("%d", x++); /* stampa 5 e poi incrementa
                      di 1 il valore di x */
x--;               /* decrementa di 1 il valore
                      di x */
printf("%d", --x); /* decrementa di 1 il valore
                      di x e poi stampa 4 */
```