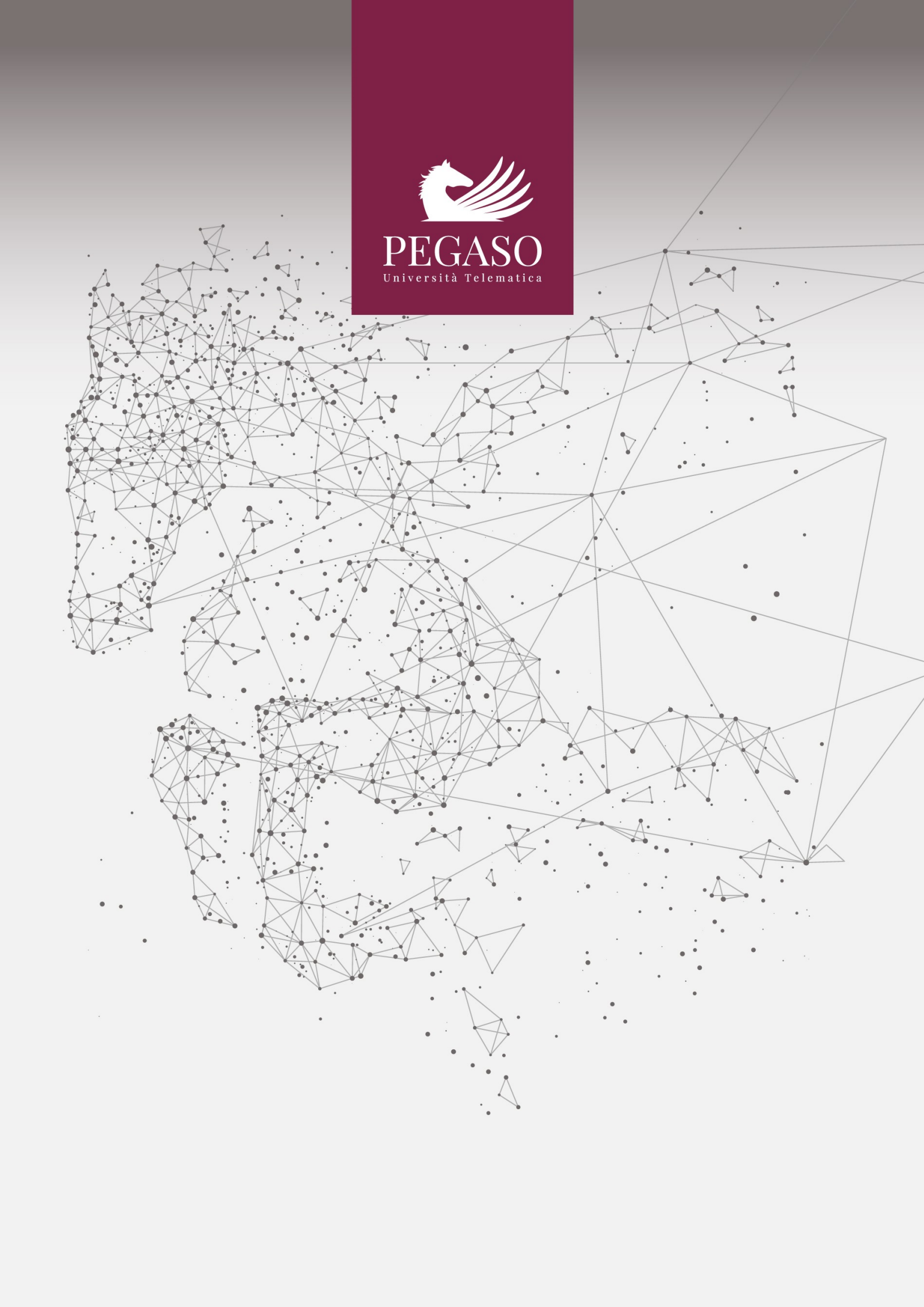

PEGASO
Università Telematica

Indice

1. Istruzione for

L'istruzione di iterazione for gestisce tutti i dettagli dell'iterazione controllata da contatore.

Per illustrare le sue potenzialità, scriviamo il programma per contare fino a 10.

Quando l'istruzione for inizia l'esecuzione, la variabile di controllo counter è inizializzata a 1.

Quindi viene controllata la condizione di continuazione del ciclo counter <= 10.

Poiché il valore iniziale di counter è 1, la condizione è soddisfatta, così l'istruzione `printf` (riga 10) stampa il valore di counter, ossia 1.

La variabile di controllo counter è poi incrementata con l'espressione `++counter` e il ciclo ricomincia con il test di continuazione del ciclo.

Poiché la variabile di controllo è ora uguale a 2, il valore finale non viene superato, così il programma esegue di nuovo l'istruzione `printf`.

Questo processo continua finché la variabile di controllo counter non è incrementata al suo valore finale di 11.

Ciò fa sì che il test di continuazione del ciclo fallisca e l'iterazione termini.

Il programma continua eseguendo la prima istruzione dopo l'istruzione `for` (in questo caso, il programma semplicemente finisce).

```
1  // Programma C
2  // Iterazione controllata da contatore con l'istruzione for.
3  #include <stdio.h>
4
5  int main( void )
6  {
7     // inizializzazione, condizione dell'iterazione e incremento
8     // sono tutti inclusi nell'intestazione dell'istruzione for.
9     for (unsigned int counter = 1; counter <= 10; ++counter) {
10       printf( "%u\n", counter );
11    }
12 }
```

```
    10      printf( "%u\n", counter );
```

```
11    }
12 }
```

2. Componenti dell'intestazione dell'istruzione for

La seguente figura esamina in maggiore dettaglio l'istruzione `for` del programma precedente.

Si noti che l'istruzione `for` "fa tutto": specifica ognuno degli elementi necessari per l'iterazione controllata da contatore con una variabile di controllo.

Se vi è più di un'istruzione nel corpo del `for`, sono necessarie le parentesi graffe per definire il corpo del ciclo (dovreste sempre inserire il corpo di un'istruzione di controllo tra parentesi graffe, anche se contiene un'unica istruzione).

☹ *Errore comune di programmazione*

Per una variabile di controllo definita nell'intestazione di un'istruzione for, il tentativo di accesso alla variabile di controllo dopo la parentesi graffa destra di chiusura (}) dell'istruzione for è un errore di compilazione.

Le variabili di controllo definite in un'intestazione del `for` esistono solo fino al termine del ciclo

Quando definite la variabile di controllo nell'intestazione del `for` prima del primo punto e virgola (;), come nella riga 9, la variabile di controllo esiste solo fino al termine del ciclo:

```
for (unsigned int counter = 1; counter <= 10; ++counter) {
```

3. Errori di tipo off-by-one

Notate che si usa la condizione di continuazione del ciclo counter <= 10.

Se per sbaglio scriveste counter < 10, il ciclo sarebbe eseguito solo 9 volte.

Questo è un comune errore logico chiamato errore di tipo off-by-one (letteralmente "sfasamento di uno").

😊 *Prevenzione di errori*

L'uso del valore finale nella condizione di un'istruzione while o di un'istruzione for assieme all'operatore relazionale <= può contribuire a evitare gli errori di tipo off-by-one. Per un ciclo usato per stampare i valori da 1 a 10, ad esempio, la condizione di continuazione del ciclo deve essere counter <= 10 invece che counter < 11 o counter < 10.

4. Formato generale di un'istruzione for

Il formato generale dell'istruzione for è

```
for (inizializzazione; condizione; incremento) {
   istruzione
}
```

dove l'espressione *inizializzazione* inizializza la variabile di controllo del ciclo (e potrebbe definirla, come abbiamo fatto precedentemente), l'espressione *condizione* è la condizione di continuazione del ciclo e l'espressione *incremento* incrementa la variabile di controllo.

Sequenze di espressioni separate da virgole

Spesso, l'espressione *inizializzazione* e l'espressione *incremento* sono sequenze di espressioni separate da virgole.

Le virgole, come sono usate qui, sono in realtà operatori virgola che garantiscono che le sequenze di espressioni vengano calcolate da sinistra a destra.

Il valore e il tipo di una sequenza di espressioni separate da virgole sono il valore e il tipo dell'espressione più a destra nella sequenza.

```
for (i = 1, t = 0; i <= 10; ++i, ++t) {
   printf( "i=%u, t=%u\n", i, t );
}
```

L'operatore virgola è molto spesso usato nell'istruzione `for`, principalmente per permettere l'uso dell'inizializzazione multipla e/o delle espressioni di incremento multiplo.

Ad esempio, in una singola istruzione `for` ci possono essere due variabili di controllo da inizializzare e incrementare.

Nelle sezioni di inizializzazione e di incremento di un'istruzione `for` mettete soltanto espressioni che riguardano le variabili di controllo.

Le manipolazioni di altre variabili devono comparire o prima del ciclo (se vengono eseguite soltanto una volta, come le istruzioni di inizializzazione) o nel corpo del ciclo (se vengono eseguite una volta per iterazione, come le istruzioni di incremento o di decremento).

Le espressioni nell'intestazione dell'istruzione `for` sono opzionali

Le tre espressioni nell'istruzione for sono opzionali.

Se viene omessa l'espressione condizione, il C considera che la condizione di continuazione del ciclo sia vera, generando così un ciclo infinito.

Potete omettere l'espressione inizializzazione se la variabile di controllo è inizializzata prima dell'istruzione `for`. L'espressione incremento può essere omessa se l'incremento è calcolato da istruzioni nel corpo dell'istruzione for o se non è necessario alcun incremento.

L'espressione incremento nell'istruzione for agisce come un'istruzione separata in C alla fine del corpo del `for`.

Quindi, le espressioni

```
counter = counter + 1
counter += 1
++counter
counter++
```

sono tutte equivalenti quando sono poste nella sezione di incremento dell'istruzione `for`. Alcuni programmatori in C preferiscono la forma `counter++` perché l'incremento avviene dopo che il corpo del ciclo è eseguito e la forma di postincremento sembra più naturale.

Poiché la variabile da preincrementare o postincrementare qui non compare in un'espressione più ampia, entrambe le forme di incremento hanno lo stesso effetto. Nell'istruzione for ci vuole due volte il punto e virgola.

😊 *Prevenzione di errori*

I cicli infiniti si verificano quando la condizione di continuazione del ciclo in un'istruzione di iterazione non diventa mai falsa.

Per evitare cicli infiniti bisogna accertarsi che non ci sia un punto e virgola immediatamente dopo l'intestazione di un'istruzione `while`*.*

In un ciclo controllato da contatore bisogna accertarsi che la variabile di controllo sia incrementata (o decrementata) nel ciclo.

In un ciclo controllato da sentinella bisogna accertarsi che il valore sentinella alla fine sia inserito.

☹ *Errore comune di programmazione*

Usare virgole invece di punti e virgola in un'intestazione `for` *è un errore di sintassi.*

5. Istruzione `for`: note e osservazioni

1. L'inizializzazione, la condizione di continuazione del ciclo e l'incremento possono contenere espressioni aritmetiche.

   Ad esempio, se x = 2 e y = 10, l'istruzione

   ```
   for (j = x; j <= 4 * x * y; j += y / x)
   ```

   è equivalente all'istruzione

   ```
   for (j = 2; j <= 80; j += 5)
   ```

2. L'incremento può essere negativo (nel qual caso, è in realtà un decremento e il ciclo conta in effetti all'indietro).
3. Se la condizione di continuazione del ciclo è inizialmente falsa, il corpo del ciclo non viene eseguito.

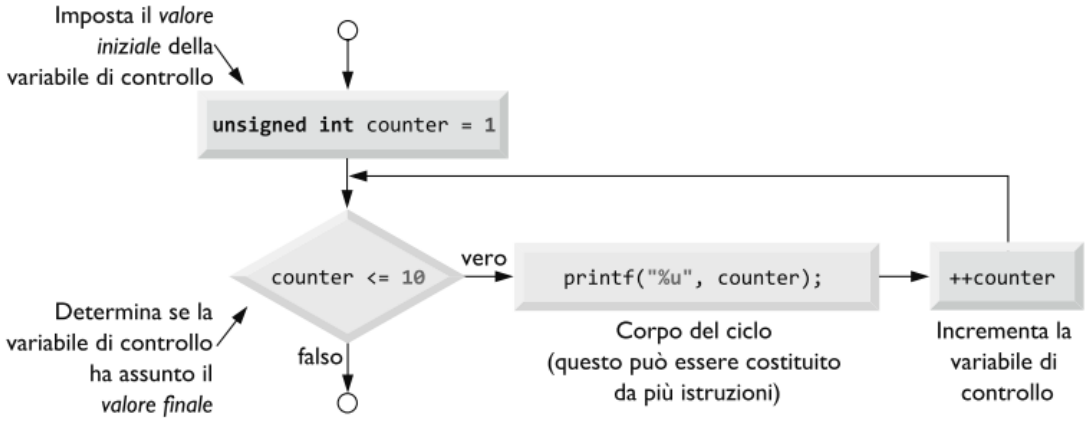
   Invece, l'esecuzione procede con l'istruzione che segue l'istruzione `for`.
4. La variabile di controllo è frequentemente stampata o usata nei calcoli all'interno del corpo di un ciclo, ma questo non è necessario.

   È comune usare la variabile di controllo per controllare l'iterazione, senza menzionarla mai nel corpo di un ciclo.
5. L'istruzione `for` è rappresentata con un diagramma di flusso allo stesso modo di un'istruzione `while`.

   Ad esempio, la seguente figura mostra il diagramma di flusso per l'istruzione `for`

```
for (unsigned int counter = 1; counter <= 10; ++counter) {
   printf("%u", counter)
}
```

Questo diagramma di flusso rende evidente che l'inizializzazione avviene solo una volta e che l'incremento è effettuato dopo ogni esecuzione dell'istruzione del corpo.



☺ *Prevenzione di errori*

Sebbene il valore della variabile di controllo si possa modificare nel corpo di un ciclo for, questo può portare a errori subdoli. È meglio non modificarlo.

6. Esempi di uso dell'istruzione `for`

I seguenti esempi mostrano alcuni metodi con cui far variare la variabile di controllo in un'istruzione `for`.

1. Far variare la variabile di controllo da 1 a 100 per incrementi di 1.

   ```
   for (unsigned int i = 1; i <= 100; ++i)
   ```

2. Far variare la variabile di controllo da 100 a 1 per incrementi di –1 (decrementi di 1).

   ```
   for (unsigned int i = 100; i >= 1; --i)
   ```

3. Far variare la variabile di controllo da 7 a 77 per incrementi di 7.

   ```
   for (unsigned int i = 7; i <= 77; i += 7)
   ```

4. Far variare la variabile di controllo da 20 a 2 per incrementi di –2.

   ```
   for (unsigned int i = 20; i >= 2; i -= 2)
   ```

5. Far variare la variabile di controllo secondo la seguente sequenza di valori: 2, 5, 8, 11, 14, 17.

   ```
   for (unsigned int j = 2; j <= 17; j += 3)
   ```

6. Far variare la variabile di controllo secondo la seguente sequenza di valori: 44, 33, 22, 11, 0.

   ```
   for (unsigned int j = 44; j >= 0; j -= 11)
   ```

☺ *Buona pratica di programmazione*

Se possibile, limitate le intestazioni delle istruzioni di controllo a una singola riga.

7. Applicazione: sommare i numeri interi pari da 2 a 100

Il seguente programma utilizza l'istruzione for per sommare tutti i numeri interi pari da 2 a 100.

Ogni iterazione del ciclo (righe 9–11) aggiunge il valore della variabile di controllo n alla variabile sum.

```
01  // Programma in C
02  // Somma con for.
03  #include <stdio.h>
04
05  int main(void)
06  {
07     unsigned int sum = 0; // inizializza sum
08
09     for (unsigned int n = 2; n <= 100; n += 2) {
10        sum += n; // aggiungi n a sum
11     }
12
13     printf("Sum is %u\n", sum);
14  }
```

```
Sum is 2550
```

8. Applicazione: calcolo dell'interesse composto

Il prossimo esempio calcola l'interesse composto con l'uso dell'istruzione `for`.

Si consideri la seguente enunciazione del problema:

Una persona investe € 1000,00 in un conto corrente che frutta il 5% di interesse.

Supponendo che l'intero interesse resti depositato nel conto, calcolate e stampate la quantità di denaro nel conto alla fine di ogni anno per 10 anni.

Usate la seguente formula per determinare queste quantità:

$a = p(1 + r)^n$

dove

p è la quantità iniziale di denaro investita (cioè il capitale)

r è il tasso annuale di interesse (ad esempio, .05 per 5%)

n è il numero degli anni

a è la quantità di denaro in deposito alla fine dell'anno *n*.

Questo problema richiede un ciclo che esegua il calcolo indicato per ognuno dei 10 anni in cui il denaro rimane in deposito.

```
1  //
2  // Calcolo dell'interesse composto.
3  #include <stdio.h>
4  #include <math.h>
5
6  int main(void)
7  {
8     double principal = 1000.0; // capitale iniziale
9     double rate = .05; // tasso di interesse annuale
10
11    // stampa le intestazioni delle colonne della tabella
```

```
12      printf("%4s%21s\n", "Year", "Amount on deposit");
13
14      // calcola la quantità in deposito per ognuno dei 10 anni
15      for (unsigned int year = 1; year <= 10; ++year) {
16
17         // calcola la nuova quantità per un anno specifico
18         double amount = principal * pow(1.0 + rate, year);
20         // stampa una riga della tabella
21         printf("%4u%21.2f\n", year, amount);
22      }
23   }
```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

L'istruzione `for` esegue il corpo del ciclo 10 volte, facendo variare la variabile di controllo da 1 a 10 per incrementi di 1.

Benché il C non includa un operatore esponenziale, possiamo usare a questo scopo la funzione `pow` (riga 18) della Libreria Standard.

La funzione `pow(x, y)` calcola il valore di `x` elevato alla potenza `y`.

Essa prende due argomenti di tipo double e restituisce un valore double.

L'intestazione `<math.h>` (riga 4) va inclusa ogni volta che si usa una funzione della libreria `math` come `pow`.

Questo programma funzionerebbe male senza l'inclusione di `math.h`, poiché il linker non sarebbe in grado di trovare la funzione `pow`.

Nota: Con diversi compilatori Linux/UNIX C dovete includere l'opzione *–lm*

(es. *gcc -o programma programma.c -lm*)

La funzione `pow` richiede due argomenti double, ma la variabile `year` è un intero.

Il file di intestazione `math.h` contiene informazioni che dicono al compilatore di convertire il valore di `year` in una rappresentazione temporanea double prima di chiamare la funzione.

Queste informazioni sono contenute nel prototipo della funzione `pow`.

Il tipo `double` è un numero in virgola mobile come `float`, ma normalmente una variabile di tipo `double` può memorizzare un valore più elevato con una precisione maggiore di `float`.

Le variabili di tipo `double` occupano più memoria di quelle di tipo `float`.

Per tutte le applicazioni eccetto quelle con uso intensivo di memoria, i programmatori professionisti generalmente preferiscono `double a float`.

Per esempio, per quantità monetarie viene considerato più opportuno l'impiego di `double`, che offre precisione maggiore.

Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione", Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.