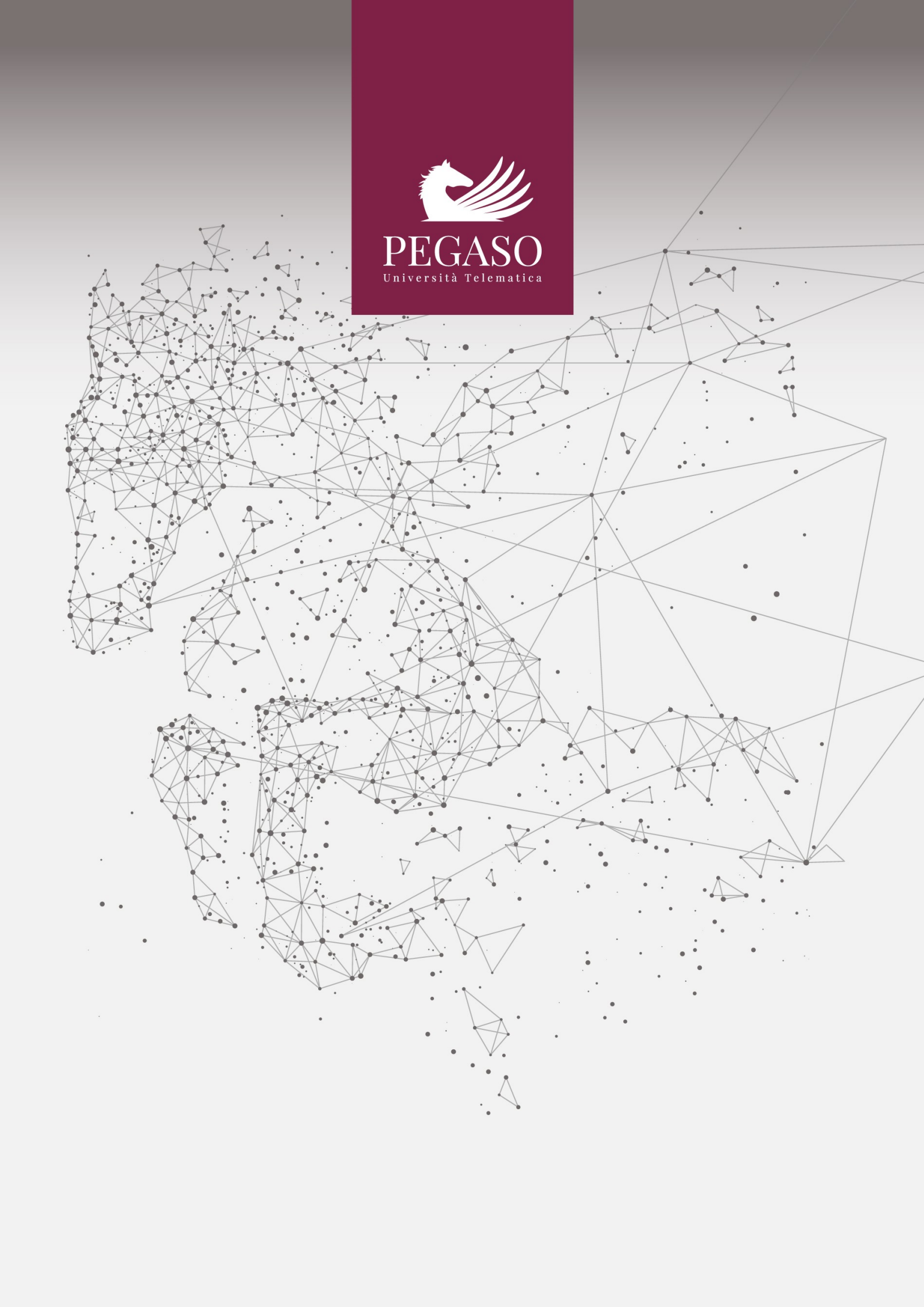
PEGASO
Università Telematica

Indice

1. Modularizzazione dei programmi in C

La maggior parte dei programmi per computer che risolvono problemi reali ha dimensioni molto più grandi rispetto ai programmi presentati fino a questo momento.

L'esperienza ha dimostrato che il modo migliore per sviluppare e mantenere un programma di grandi dimensioni è quello di costruirlo partendo da elementi più piccoli, ognuno dei quali è più maneggevole del programma originario (dividi e conquista).

Introdurremo alcune caratteristiche chiave del linguaggio C che facilitano la progettazione, l'implementazione, l'utilizzo e la manutenzione di programmi di grandi dimensioni.

In C si utilizzano le funzioni per la modularizzazione dei programmi.

I programmi sono tipicamente scritti combinando la scrittura di nuove funzioni con l'utilizzazione di funzioni preconfezionate disponibili nella Libreria Standard del C.

La Libreria Standard del C fornisce una ricca collezione di funzioni per eseguire comuni calcoli matematici, manipolazioni di stringhe, manipolazioni di caratteri, input/output e molte altre operazioni utili.

☺ *Buona pratica di programmazione*

Acquisite familiarità con la ricca collezione di funzioni della Libreria Standard del C

☺ *Osservazione di ingegneria del software*

Evitate di reinventare la ruota. Quando è possibile, usate le funzioni della Libreria Standard del C invece di scrivere nuove funzioni. Ciò permette di ridurre il tempo di sviluppo dei programmi. Queste funzioni sono scritte da esperti, ben testate ed efficienti.

☺ *Portabilità*

L'uso delle funzioni della Libreria Standard del C contribuisce a rendere i programmi più portabili.

Il linguaggio C e la Libreria Standard sono entrambi specificati dal C standard, ed entrambi sono forniti dai sistemi conformi al C standard (con l'eccezione di alcune librerie che sono considerate opzionali).

Le funzioni `printf e scanf`, sono funzioni della Libreria Standard.

Potete scrivere funzioni per definire compiti specifici, che possono essere usate in molti punti di un programma.

Queste sono chiamate funzioni definite dal programmatore.

Le particolari istruzioni che definiscono una funzione sono scritte una sola volta e sono nascoste alle altre funzioni.

Una funzione è *invocata* da una *chiamata di funzione*, che specifica il nome della funzione e fornisce le informazioni (come *argomenti*) di cui la funzione ha bisogno per eseguire il suo compito designato.

Qualcosa di analogo a ciò è la forma gerarchica del management.

Una dirigente (la funzione che chiama o chiamante) chiede a un'impiegata esecutrice (la funzione chiamata) di eseguire un compito e di fare rapporto quando il compito è eseguito.

Ad esempio, una funzione che deve stampare informazioni sullo schermo chiama la funzione esecutrice `printf` per eseguire quel compito, quindi `printf` stampa le informazioni e, quando il suo compito è completato, riferisce (o torna) alla funzione chiamante.

La funzione "dirigente" non sa come la funzione "esecutrice" esegue i suoi compiti.

La funzione esecutrice può chiamare altre funzioni esecutrici e la dirigente può essere ignara di ciò.

"Nascondere" i dettagli di implementazione favorisce una buona ingegneria del software.

Le relazioni tra le funzioni possono essere anche di natura diversa rispetto alla struttura gerarchica.

2. Introduzione alle funzioni

Le funzioni permettono di rendere modulare un programma.

Tutte le variabili definite nelle definizioni di funzione sono variabili locali: è possibile accedervi solo all'interno della funzione nella quale sono definite.

La maggior parte delle funzioni ha una lista di parametri che permettono la comunicazione di informazioni tra funzioni tramite argomenti nelle chiamate delle funzioni.

I parametri di una funzione sono anche variabili locali di quella funzione.

😊 Osservazione di ingegneria del software

In programmi contenenti molte funzioni, `main` è spesso implementata come un insieme di chiamate a funzioni che effettuano il grosso del lavoro del programma.

Vi sono svariate motivazioni per "funzionalizzare" un programma.

L'approccio dividi e conquista rende più fattibile lo sviluppo del programma.

Un'altra motivazione sta nella riusabilità del software, poiché si possono usare le funzioni esistenti come blocchi costituenti per creare nuovi programmi.

Con una buona scelta dei nomi e delle definizioni delle funzioni si possono creare programmi partendo da funzioni standardizzate che eseguono compiti specifici, piuttosto che costruirli usando un codice sviluppato personalmente.

Ciò è noto come astrazione.

Usiamo l'astrazione ogni volta che usiamo funzioni della Libreria Standard come `printf`, `scanf` e `pow`.

Una terza motivazione sta nell'evitare di ripetere il codice in un programma. Impacchettare il codice come funzione ne permette l'esecuzione in altri punti del programma semplicemente chiamando la funzione.

😊 *Osservazione di ingegneria del software*

Ogni funzione deve limitarsi a eseguire un singolo compito ben definito e il nome della funzione deve esprimere quel compito.

Questo facilita l'astrazione e favorisce la riutilizzabilità del software.

😊 *Osservazione di ingegneria del software*

Se non sapete scegliere un nome conciso che indichi il compito della funzione, è possibile che la vostra funzione stia tentando di eseguire troppi compiti diversi.

Di solito, è meglio spezzare una funzione di questo tipo in varie funzioni più piccole.

Questa operazione è detta scomposizione.

libreria math

Le funzioni della libreria `math` permettono di eseguire certi comuni calcoli matematici. Ne useremo qui alcune al fine di introdurre il concetto di funzione.

Nel seguito esamineremo molte altre funzioni della Libreria Standard del C.

Le funzioni sono usate normalmente in un programma scrivendo il nome della funzione seguito da una parentesi sinistra, seguita poi dall'argomento (o da un elenco di argomenti separati da virgola) della funzione, seguito infine da una parentesi destra.

Ad esempio, per calcolare e stampare la radice quadrata di 900.0 potete scrivere

```
printf("%.2f", sqrt(900.0));
```

Quando questa istruzione viene eseguita, la funzione della libreria `math sqrt` è chiamata a calcolare la radice quadrata del numero contenuto entro le parentesi (900.0). Il numero 900.0 è l'argomento della funzione `sqrt`.

L'istruzione precedente stampa 30.00.

La funzione `sqrt` riceve un argomento di tipo `double` e restituisce un risultato di tipo `double`.

Tutte le funzioni nella libreria `math` che restituiscono valori in virgola mobile restituiscono il tipo di dati `double`.

Si noti che i valori `double`, come i valori `float`, possono essere stampati usando la specificazione di conversione `%f`.

Potete anche memorizzare il risultato di una chiamata di una funzione in una variabile

per un uso successivo come in:

```
double result = sqrt(900.0);
```

😊 *Prevenzione di errori*

Quando usate le funzioni della libreria `math`, includete l'intestazione `math.h` usando la direttiva del preprocessore #include `<math.h>`

Gli argomenti di una funzione possono essere costanti, variabili o espressioni.

Se `c1 = 13.0, d = 3.0 e f = 4.0`, l'istruzione

```
printf("%.2f", sqrt(c1 + d * f));
```

calcola e stampa la radice quadrata di 13.0 + 3.0 * 4.0 = 25.0, cioè 5.00.

La seguente figura riassume alcune delle funzioni della libreria math del C.

Nella figura le variabili x e y sono di tipo double.

Funzione	Descrizione	Esempio
`sqrt(x)`	radice quadrata di *x*	`sqrt(900.0)` è uguale a 30.0 `sqrt(9.0)` è uguale a 3.0
`cbrt(x)`	radice cubica di *x* (solo per il C99 e il C11)	`cbrt(27.0)` è uguale a 3.0 `cbrt(-8.0)` è uguale a -2.0
`exp(x)`	funzione esponenziale e^x	`exp(1.0)` è uguale a 2.718282 `exp(2.0)` è uguale a 7.389056
`log(x)`	logaritmo naturale di *x* (in base *e*)	`log(2.718282)` è uguale a 1.0 `log(7.389056)` è uguale a 2.0
`log10(x)`	logaritmo di *x* (in base 10)	`log10(1.0)` è uguale a 0.0 `log10(10.0)` è uguale a 1.0 `log10(100.0)` è uguale a 2.0
`fabs(x)`	valore assoluto di *x* come numero in virgola mobile	`fabs(13.5)` è uguale a 13.5 `fabs(0.0)` è uguale a 0.0 `fabs(-13.5)` è uguale a 13.5
`ceil(x)`	arrotonda *x* all'intero più piccolo non minore di *x*	`ceil(9.2)` è uguale a 10.0 `ceil(-9.8)` è uguale a -9.0
`floor(x)`	arrotonda *x* all'intero più grande non maggiore di *x*	`floor(9.2)` è uguale a 9.0 `floor(-9.8)` è uguale a -10.0
`pow(x, y)`	*x* elevato alla potenza *y* (x^y)	`pow(2, 7)` è uguale a 128.0 `pow(9, .5)` è uguale a 3.0
`fmod(x, y)`	resto di *x*/*y* come numero in virgola mobile	`fmod(13.657, 2.333)` è uguale a 1.992
`sin(x)`	funzione trigonometrica seno di *x* (*x* in radianti)	`sin(0.0)` è uguale a 0.0
`cos(x)`	funzione trigonometrica coseno di *x* (*x* in radianti)	`cos(0.0)` è uguale a 1.0
`tan(x)`	funzione trigonometrica tangente di *x* (*x* in radianti)	`tan(0.0)` è uguale a 0.0

Lo standard C11 aggiunge un'ampia gamma di funzionalità in virgola mobile e per operare con i numeri complessi.

3. Esercitazioni

Esempio 1

$$y = \frac{ax^3 - 3x^2}{2ab}$$

L'espressione si scrive con l'istruzione di assegnazione seguente:

```
y = (a* pow(x,3) - 3*pow(x,5)) / (2*a*b)
```

oppure

```
y = (a* pow(x,3) - 3*pow(x,5)) /2/a/b
```

Esempio 2

$$z = \frac{\sqrt{x^4 - 3}}{|x - 1|}$$

L'espressione si scrive con l'istruzione di assegnazione seguente:

```
z = sqrt( pow(x,4) - 3 ) / fabs( x-1 )
```

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.