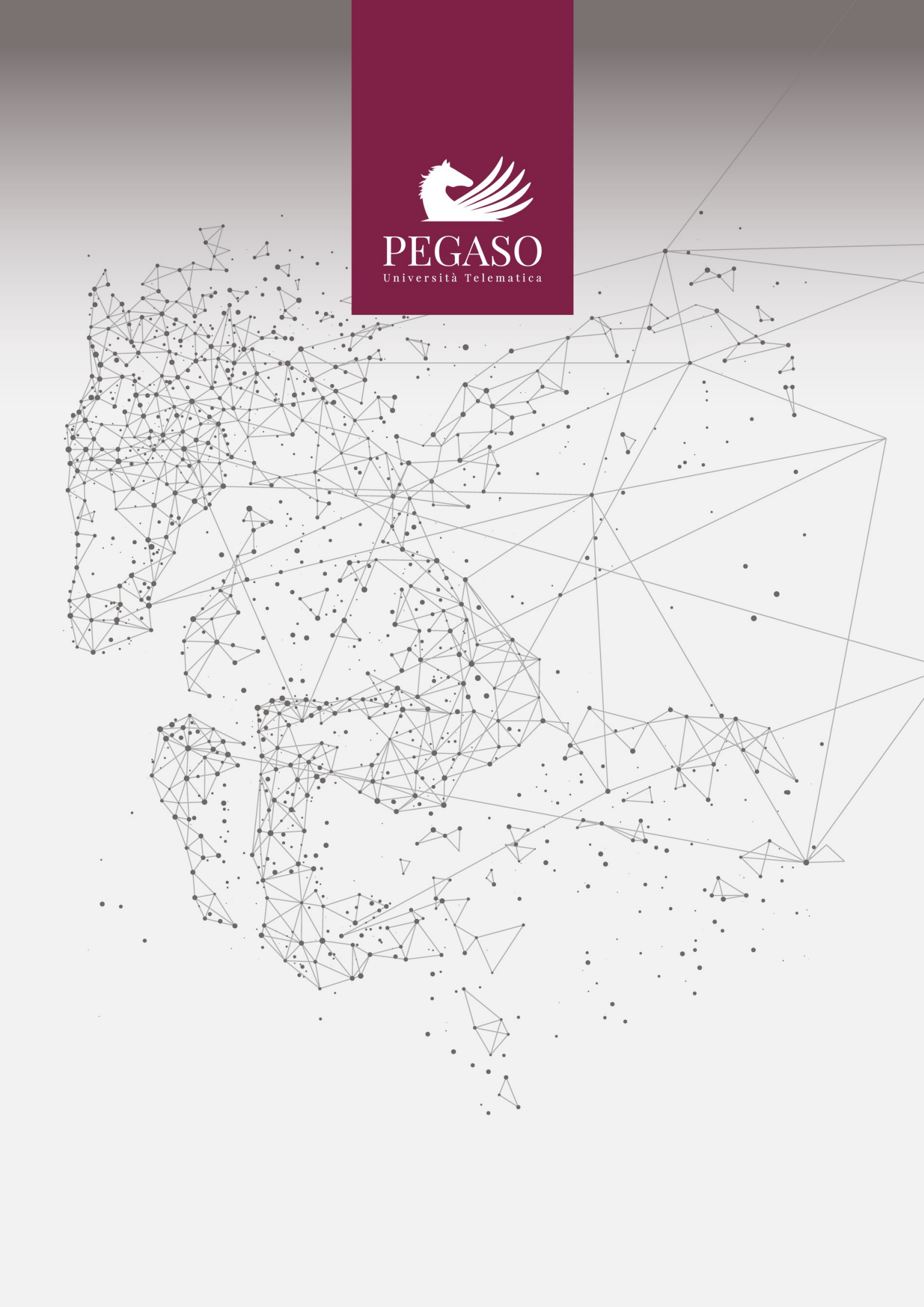
PEGASO
Università Telematica

Indice

1. Esempio: La funzione maximum

Il nostro secondo esempio usa una funzione `maximum` definita dal programmatore per identificare e restituire il maggiore di tre numeri interi.

Gli interi sono letti con `scanf` (riga 14), poi vengono passati a `maximum` (riga 18), che trova l'intero maggiore.

Questo valore è restituito a `main` con l'istruzione `return` in `maximum` (riga 35).

L'istruzione `printf` nella riga 18 stampa quindi il valore restituito da `maximum`.

```
01  // Programma C
02  // Trovare il massimo di tre interi.
03  #include <stdio.h>
04
05  int maximum(int x, int y, int z); // prototipo di funzione
06
07  int main(void)
08  {
09     int num1; // primo intero inserito dall'utente
10     int num2; // secondo intero inserito dall'utente
11     int num3; // terzo intero inserito dall'utente
12
13     printf("%s", "Enter three integers: ");
14     scanf("%d%d%d", &num1, &num2, &num3);
15
16     // num1, num2 e num3 sono argomenti
17     // nella chiamata della funzione maximum
18     printf("Maximum is: %d\n", maximum(num1, num2, num3));
19  }
20
21  // definizione della funzione maximum
```

```
22  // x, y e z sono parametri
23  int maximum(int x, int y, int z)
24  {
25     int max = x; //supponi che x sia il maggiore
26
27     if (y > max) { // se y e' più grande di max,
28        max = y; // assegna y a max
29     }
30
31     if (z > max) { // se z e' più grande di max,
32        max = z; // assegna z a max
33     }
34
35     return max; // max e' il valore più grande
36  }
```

```
Enter three integers: 22 85 17

Maximum is: 85
```

```
Enter three integers: 47 32 14

Maximum is: 47
```

```
Enter three integers: 35 8 79

Maximum is: 79
```

La funzione inizialmente presuppone che il suo primo argomento (memorizzato nel parametro `x`) sia il maggiore e lo assegna a `max` (riga 25).

Successivamente, l'istruzione `if` alle righe 27–29 determina se `y` è maggiore di `max` e, in caso affermativo, assegna `y` a `max`.

Poi, l'istruzione `if` alle righe 31–33 determina se `z` è maggiore di `max` e, in caso affermativo, assegna `z` a `max`.

Infine, la riga 35 restituisce `max` alla funzione chiamante.

Approfondimento su prototipi di funzioni

Un'importante caratteristica del C è il prototipo di una funzione.

Il compilatore usa i prototipi di funzione per convalidare le chiamate delle funzioni.

Le versioni precedenti del C non eseguivano questo genere di controllo, così era possibile chiamare le funzioni in maniera impropria senza che il compilatore individuasse gli errori.

Simili chiamate potevano produrre errori irreversibili al momento dell'esecuzione o errori non irreversibili che causavano problemi indefinibili, difficili da scoprire.

I prototipi di funzione correggono questa mancanza.

☺ *Buona pratica di programmazione*

Includete i prototipi di funzione per tutte le funzioni, così da trarre vantaggio dalle capacità di controllo dei tipi del C.

Usate le direttive per il preprocessore #include *per ottenere dai file di intestazione relativi alle librerie utilizzate i prototipi di funzione per le funzioni della Libreria Standard, o per includere le intestazioni contenenti i prototipi di funzione per le funzioni sviluppate da voi e/o dai membri del vostro gruppo.*

Il prototipo di funzione per maximum (riga 5) è

```
int maximum(int x, int y, int z); // prototipo di funzione
```

Esso stabilisce che maximum riceve tre argomenti di tipo `int` e restituisce un risultato di tipo `int`.

Notate che il prototipo di funzione coincide con la prima riga della definizione di `maximum`.

😊 *Buona pratica di programmazione*

Includete i nomi dei parametri nei prototipi di funzione per scopi di documentazione. Il compilatore ignora questi nomi, quindi il prototipo int maximum(int, int, int); è valido..

☹ *Errore comune di programmazione 5.5*

Dimenticare il punto e virgola alla fine del prototipo di una funzione è un errore di sintassi.

Errori di compilazione

La chiamata di una funzione che non si accorda con il prototipo della funzione stessa genera un errore di compilazione.

Viene generato un errore anche se non concordano tra loro il prototipo e la definizione della funzione.

Ad esempio, se il prototipo di funzione fosse scritto

```
void maximum(int x, int y, int z);
```

il compilatore genererebbe un errore, perché il tipo di ritorno `void` nel prototipo di funzione sarebbe stato diverso dal tipo di ritorno `int` dell'intestazione della funzione.

Coercizione degli argomenti e "normali regole di conversione aritmetica"

Un'altra importante caratteristica dei prototipi di funzione è la coercizione degli argomenti, ossia la forzatura nei confronti degli argomenti perché essi siano del tipo appropriato.

Ad esempio, la funzione della libreria `math sqrt` può essere chiamata con un argomento intero anche se il prototipo della funzione in `<math.h>` specifica un parametro double; anche in questo caso la funzione opererà correttamente.

L'istruzione

```
printf("%.3f\n", sqrt(4));
```

calcola correttamente `sqrt(4)` e stampa il valore 2.000.

Il prototipo della funzione fa sì che il compilatore converta una copia del valore `int 4` nel valore `double` 4.0 prima che tale copia sia passata a `sqrt`.

In generale, i valori degli argomenti che non corrispondono precisamente ai tipi dei parametri nel prototipo della funzione vengono convertiti nel tipo adatto prima che la funzione sia chiamata.

Queste conversioni possono portare a risultati scorretti se non si seguono le normali regole di conversione aritmetica del C.

Queste regole specificano come i valori possono essere convertiti in altri tipi senza perdere dati.

Nel nostro esempio relativo a `sqrt`, un `int` viene convertito automaticamente in un double senza che venga modificato il suo valore (perché `double` può rappresentare un intervallo di valori molto più grande di `int`).

Tuttavia, la conversione di un `double` in un `int` fa sì che venga troncata la parte frazionaria di `double`, modificando così il valore originario.

Convertire tipi interi più grandi in tipi interi più piccoli (es. `long` in `short`) può anche produrre valori modificati.

Le normali regole di conversione aritmetica si applicano automaticamente alle espressioni contenenti valori di due tipi di dati diversi (dette anche espressioni con tipi misti) e sono gestite dal compilatore.

In un'espressione con tipi misti il compilatore fa una copia temporanea del valore che deve essere convertito, quindi converte la copia nel tipo "più alto" nell'espressione (questo è noto come promozione).

Le normali regole di conversione aritmetica per un'espressione di tipo misto, contenente almeno un valore in virgola mobile, sono:

- Se uno dei valori è un `long double`, l'altro si converte in un `long double`.
- Se uno dei valori è un `double`, l'altro si converte in un `double`.

• Se uno dei valori è un `float`, l'altro si converte in un `float`.

Se l'espressione con tipi misti contiene solo tipi interi, le normali conversioni aritmetiche specificano un insieme di regole di promozione intera.

Nella maggior parte dei casi i tipi interi più in basso si convertono nei tipi più in alto.

La figura seguente elenca i tipi di dati in virgola mobile e interi con le specificazioni di conversione di ogni tipo per `printf e scanf`.

Tipo di dati	Specificazione di conversione per printf	Specificazione di conversione per scanf
Tipi in virgola mobile		
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
Tipi interi		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

Un valore si può convertire in un tipo più basso solamente assegnando esplicitamente il valore a una variabile di tipo più basso, o usando un operatore `cast`.

Gli argomenti nella chiamata di una funzione si convertono nei tipi dei parametri specificati nel prototipo della funzione, come se gli argomenti fossero assegnati direttamente a variabili di quei tipi.

Se una nostra funzione `square` che realizza il quadrato di un numero utilizzando un parametro `int` è chiamata con argomento in virgola mobile, l'argomento viene convertito in un `int` (un tipo più basso) e `square` normalmente restituisce un valore scorretto.

Ad esempio, `square(4.5)` restituisce 16, non 20.25.

☹ *Errore comune di programmazione*

Convertire da un tipo di dati più alto nella gerarchia di promozione in un tipo più basso può cambiare il valore dei dati. In tali casi molti compilatori emettono avvisi.

Se non vi è alcun prototipo per una funzione, il compilatore forma il proprio la prima volta che incontra la funzione, ossia o la sua definizione o una sua chiamata.

Ciò tipicamente fa emettere messaggi di avviso o messaggi di errore, a seconda del compilatore.

😊 *Prevenzione di errori*

Includete sempre i prototipi di una funzione per le funzioni che definite o usate nel vostro programma, in modo da prevenire errori e avvisi al momento della compilazione.

Osservazione di ingegneria del software

Il prototipo di una funzione posto al di fuori della definizione di qualsiasi altra funzione si rivolge a tutte le chiamate della funzione che compaiono nel file dopo il prototipo della funzione.

Il prototipo di una funzione posto all'interno di un'altra funzione si rivolge soltanto alle chiamate fatte in quella funzione.

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.