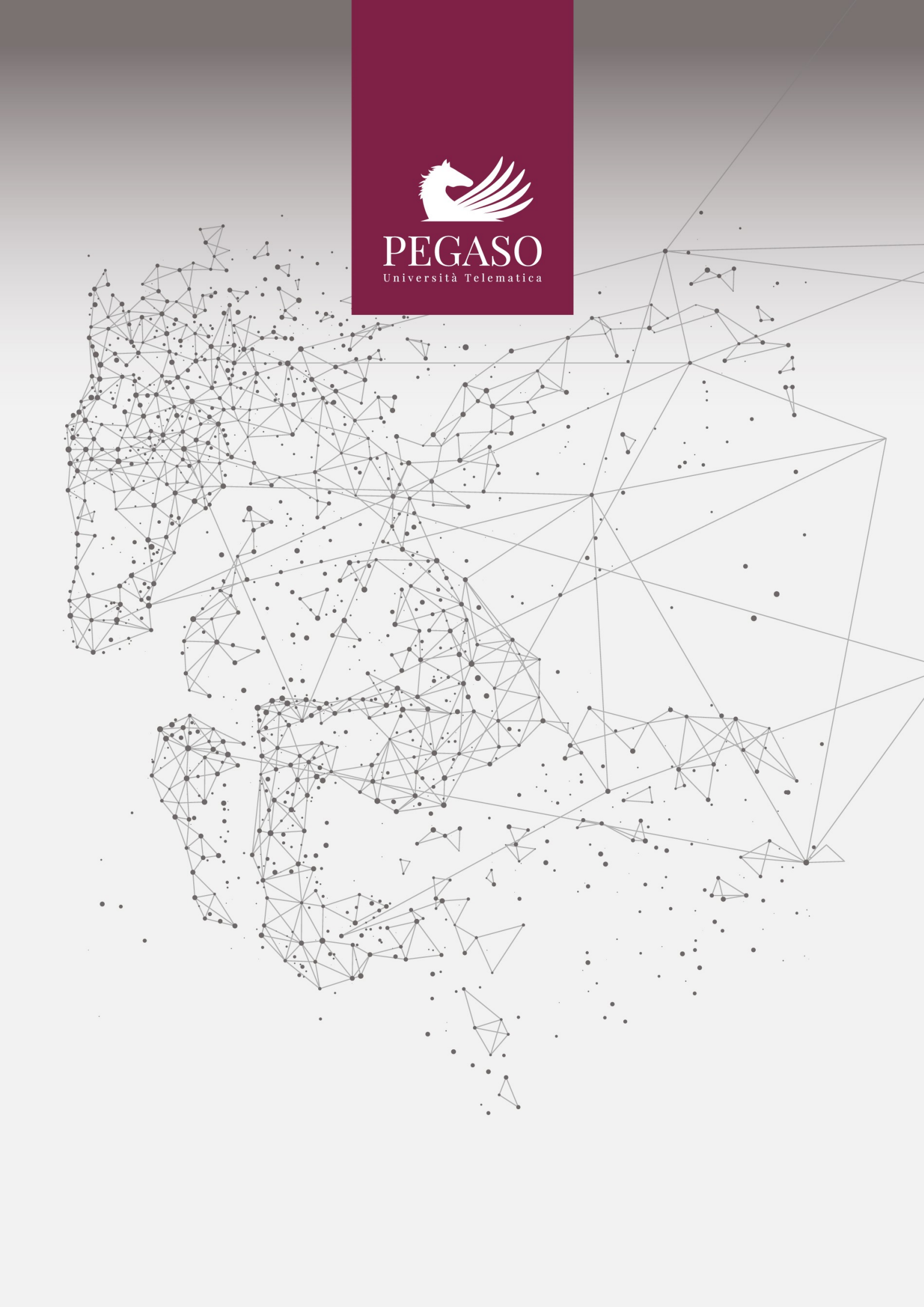
PEGASO
Università Telematica

Indice

1. Operatori logici

Finora abbiamo studiato soltanto condizioni semplici, come counter <= 10, total > 1000, e number != sentinella.

Abbiamo espresso queste condizioni in termini di operatori relazionali >, <, >= e <= e di operatori di uguaglianza == e !=.

Ogni decisione verificava precisamente una sola condizione.

Per verificare varie condizioni nel processo decisionale, dovevamo eseguire questi test in istruzioni separate o in istruzioni annidate if o if...else.

Il C fornisce operatori logici che si possono usare per formare condizioni più complesse combinando condizioni semplici.

Gli operatori logici sono && (AND logico), || (OR logico) e ! (NOT logico, chiamato anche negazione logica).

2. Operatore logico AND (&&)

Supponiamo di volerci assicurare che due condizioni siano entrambe vere prima di scegliere un certo percorso di esecuzione.

In questo caso, possiamo usare l'operatore logico && come segue:

```
if (gender == 1 && age >= 65) {
    ++seniorFemales;
}
```

Questa istruzione `if` contiene due condizioni semplici.

La condizione `gender == 1` potrebbe essere valutata, ad esempio, per determinare se una persona è di sesso femminile.

La condizione `age >= 65` è valutata per determinare se una persona è un cittadino anziano.

Le due condizioni semplici sono valutate prima, perché == e >= hanno una precedenza più alta di &&.

L'istruzione `if` quindi considera la condizione combinata

```
gender == 1 && age >= 65
```

che è vera se e solo se entrambe le condizioni semplici sono vere.

Infine, se questa condizione combinata è vera, allora il contatore seniorFemales è incrementato di 1.

Se una delle due o entrambe le condizioni semplici sono false, allora il programma salta l'incremento e procede all'istruzione che segue l'`if`.

La tabella riepiloga l'operatore &&, mostrando tutte e quattro le possibili combinazioni dei valori zero (falso) e non zero (vero) per le espressioni costituenti espressione1 ed espressione2.

Tali tabelle sono spesso chiamate tabelle di verità.

Il C assegna a tutte le espressioni che includono operatori relazionali, operatori di uguaglianza e/o operatori logici il valore 0 o 1.

Benché il C assegni 1 a un valore vero, esso accetta come vero qualsiasi valore diverso da zero.

espressione1	**espressione2**	**espressione1 && espressione2**
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

3. Operatore logico OR (||)

Consideriamo adesso l'operatore logico `||` (OR logico).

Supponiamo di volerci assicurare, a un certo punto di un programma, che di due condizioni una o entrambe siano vere prima di scegliere un certo percorso di esecuzione. In questo caso, usiamo l'operatore `||`, come nel seguente segmento di programma:

```
if (semesterAverage >= 90 || finalExam >= 90) {
    puts("Student grade is A");
}
```

Anche questa istruzione contiene due condizioni semplici.

La condizione

```
semesterAverage >= 90
```

è calcolata per determinare se lo studente meriti una "A" come voto del corso per un rendimento eccellente in tutto il semestre.

La condizione

```
finalExam >= 90
```

è calcolata per determinare se lo studente meriti una "A" come voto del corso per un'eccezionale prestazione all'esame finale.

L'istruzione `if` considera quindi la condizione combinata

```
semesterAverage >= 90 || finalExam >= 90
```

e assegna allo studente una "A" se una o entrambe le condizioni semplici sono vere.

Il messaggio "Student grade is A" non è stampato solo quando entrambe le condizioni semplici sono false (zero). .

La tabella di verità per l'operatore logico OR (`||`) risulta quindi essere:

espressione1	**espressione2**	**espressione1 || espressione2**
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

4. Valutazione cortocircuitata

L'operatore && ha una precedenza più alta di `||`.

Entrambi gli operatori sono associativi da sinistra a destra.

Un'espressione contenente gli operatori && o || è calcolata soltanto finché non sia nota la verità o la falsità.

Così la valutazione della condizione

```
gender == 1 && age >= 65
```

si arresterà se gender non è uguale a 1 (cioè se l'intera espressione è falsa) e continuerà se gender è uguale a 1 (poiché l'intera espressione potrebbe ancora essere vera se age >= 65).

Questa caratteristica riguardante le prestazioni per la valutazione delle espressioni logiche AND e OR è chiamata valutazione cortocircuitata.

5. Prestazioni

Nelle espressioni che usano l'operatore &&, fate in modo che la condizione con maggiori probabilità di essere falsa sia quella più a sinistra.

Nelle espressioni che usano l'operatore || fate in modo che la condizione con maggiori probabilità di essere vera sia quella più a sinistra.

Questo può ridurre il tempo di esecuzione di un programma.

6. Operatore logico di negazione (!)

Il C fornisce l'operatore ! (negazione logica) per permettervi di "invertire" il significato di una condizione.

Diversamente dagli operatori && e ||, che combinano due condizioni (e sono quindi operatori binari), l'operatore logico di negazione ha soltanto una condizione singola come operando (ed è pertanto un operatore unario).

L'operatore logico di negazione è posto prima di una condizione, quando siamo interessati a scegliere un percorso di esecuzione se la condizione originaria (senza l'operatore logico di negazione) è falsa, come nel seguente segmento di programma:

```
if (!(grade == sentinelValue)) {
    printf("The next grade is %f\n", grade);
}
```

Le parentesi attorno alla condizione grade == sentinelValue sono necessarie perché l'operatore logico di negazione ha una precedenza più alta dell'operatore di uguaglianza.

Di seguito la tabella di verità per l'operatore logico di negazione.

espressione	!espressione
0	1
nonzero	0

Nella maggior parte dei casi potete evitare di usare la negazione logica, esprimendo la condizione in modo diverso con un appropriato operatore relazionale.

Ad esempio, l'istruzione precedente si può anche scrivere come segue:

```
if (grade != sentinelValue) {
    printf("The next grade is %f\n", grade);
}
```

7. Riepilogo della precedenza e dell'associatività degli operatori

La seguente tabella mostra la precedenza e l'associatività degli operatori introdotti fino a questo punto.

Gli operatori sono mostrati dall'alto in basso in ordine decrescente di precedenza.

Operatori	Associatività	Tipo
++ *(postfisso)* -- *(postfisso)*	da destra a sinistra	postfisso
+ - ! ++ *(prefisso)* -- *(prefisso)* *(tipo)*	da destra a sinistra	unario
* / %	da sinistra a destra	moltiplicativo
+ -	da sinistra a destra	additivo
< <= > >=	da sinistra a destra	relazionale
== !=	da sinistra a destra	di uguaglianza
&&	da sinistra a destra	AND logico
||	da sinistra a destra	OR logico
?:	da destra a sinistra	condizionale
= += -= *= /= %=	da destra a sinistra	di assegnazione
,	da sinistra a destra	virgola

8. Leggi di De Morgan

Le Leggi di De Morgan possono talvolta renderci più conveniente esprimere un'espressione logica.

Queste leggi stabiliscono che l'espressione

!(condizione1 && condizione2)

è logicamente equivalente all'espressione

(!condizione1 || !condizione2).

Inoltre, l'espressione

!(condizione1 || condizione2)

è logicamente equivalente all'espressione

(!condizione1 && !condizione2).

Usiamo le Leggi di De Morgan per scrivere espressioni equivalenti per ognuna delle seguenti espressioni, e poi scriviamo un programma per mostrare che sia l'espressione originaria che l'espressione nuova sono equivalenti in ciascun caso.

a) !(x < 5) && !(y >= 7)

b) !(a == b) || !(g != 5)

c) !((x <= 8) && (y > 4))

d) !((i > 4) || (j <= 6))

```
#include <stdio.h>
int main(void)
{
   int x = 10; // definisci il valore della variabile corrente
   int y = 1; // definisci il valore della variabile corrente
   int a = 3; // definisci il valore della variabile corrente
   int b = 3; // definisci il valore della variabile corrente
```

```
int g = 5; // definisci il valore della variabile corrente
int Y = 1; // definisci il valore della variabile corrente
int i = 2; // definisci il valore della variabile corrente
int j = 9; // definisci il valore della variabile corrente

// stampa i valori delle variabili
puts("current variable values are: ");
printf("x = %d, y = %d, a = %d,", x, y, a);
printf(" b = %d\n", b);
printf("g = %d, Y = %d, i = %d,", g, Y, i);
printf(" j = %d\n\n", j);

// parte a
puts( "!(x < 5) && !(y >= 7)" );
if ((!(x < 5) && !(y >= 7)) == (!((x < 5) || (y >= 7)))) {
    puts( "is equivalent to" );
} else {
    puts("is NOT equivalent to" );
}
puts( "!((x < 5) || (y >= 7))" );

// parte b
puts( "!(a == b) || !(g != 5) ");
if ((!(a == b) || !(g != 5)) == (!((a == b) && (g != 5) ))) {
    puts( "is equivalent to");
} else {
    puts( "is NOT equivalent to");
}
puts( "!((a == b) && (g != 5))");

// parte c
puts( "!((x <= 8) && (Y > 4))" );
```

```
    if (!((x <= 8) && (Y > 4)) == (!(x <= 8) || !(Y > 4))) {
       puts( "is equivalent to" );
    } else {
       puts( "is NOT equivalent to" );
    }
    puts( "(!(x <= 8) || !(Y > 4))" );

    // parte d
    puts("!((i > 4) || (j <= 6))" );
    if (!((i > 4) || (j <= 6)) == (!(i > 4) && !(j <= 6) )) {
        puts("is equivalent to");
    } else {
        puts("is NOT equivalent to");

        puts( "(!(i > 4) && !(j <= 6))" );
    }
}
```

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.