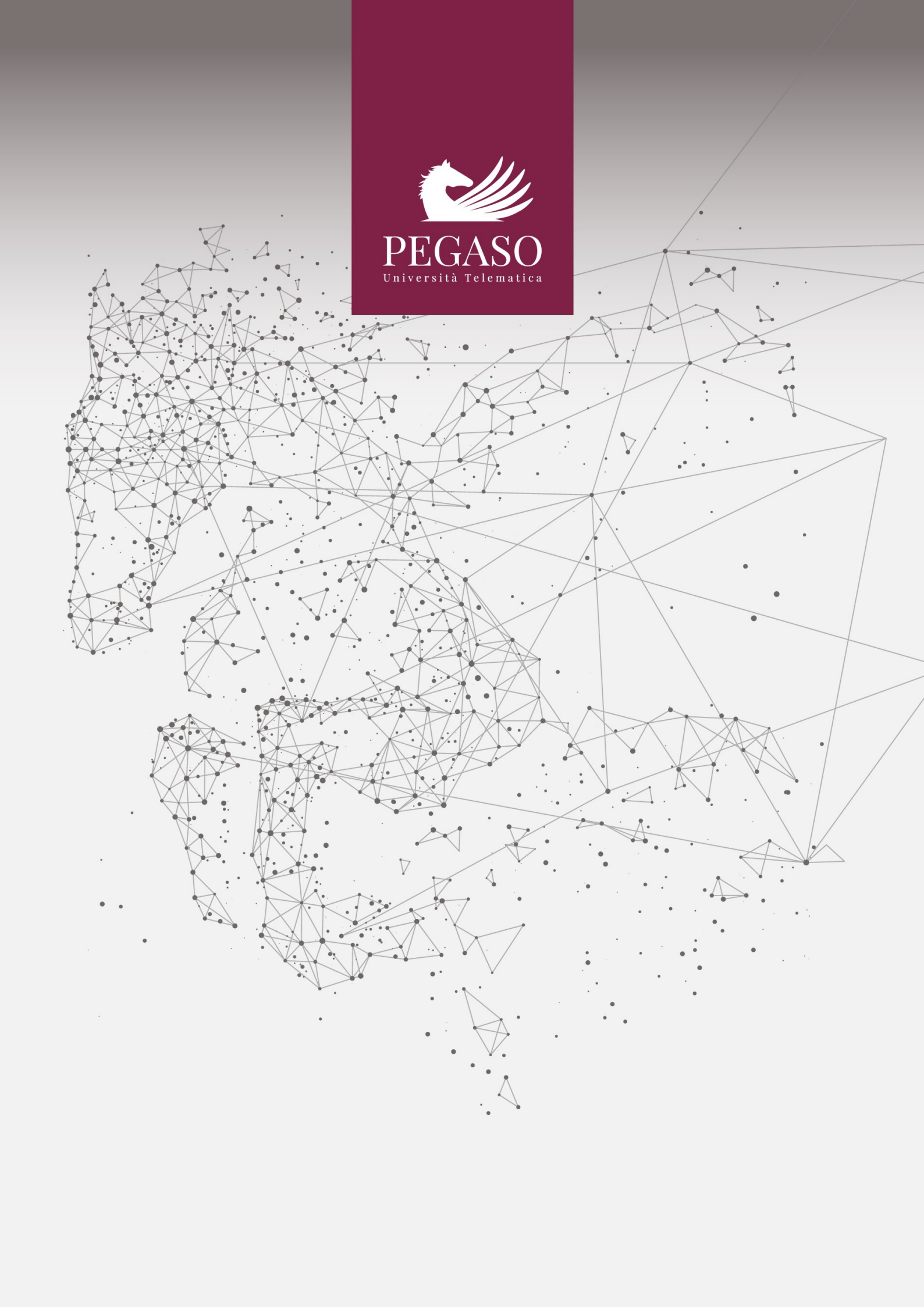
PEGASO
Università Telematica

Indice

1. FLOAT E CONVERSIONI FRA TIPI 3
2. OPERATORI DI ASSEGNAZIONE, DI INCREMENTO E DI DECREMENTO 10
3. OVERFLOW ARITMETICO - PROGRAMMAZIONE SICURA IN C 17
BIBLIOGRAFIA 20

1. Float e conversioni fra tipi

```
01  // 10_01.c
02  // Media di una classe con iterazione controllata da sentinella.
03  #include <stdio.h>
04
05  // la funzione main inizia l'esecuzione del programma
06  int main( void )
07  {
08     unsigned int counter; // numero di voti letti
09     int grade; // valore del voto
10     int total; // somma dei voti
11
12     float average; // numero con punto decimale per la media
13
14     // fase di inizializzazione
15     total = 0; // inizializza il totale
16     counter = 0; // inizializza il contatore del ciclo
17
18     // fase di elaborazione
19     // ricevi il primo voto dall'utente
20     printf( "%s", "Enter grade, -1 to end: " ); // prompt per l'input
21     scanf( "%d", &grade ); // leggi il voto dall'utente
22
23     // ripeti finche' non viene letto il valore sentinella
24     while ( grade != -1 ) {
```

```
25      total = total + grade; // aggiungi il voto al totale
26      counter = counter + 1; // incrementa il contatore
27
28      // ricevi il voto successivo dall'utente
29      printf( "%s", "Enter grade, -1 to end: " ); // prompt
30      scanf("%d", &grade); // leggi il prossimo voto
31    } // fine di while
32
33    // fase di chiusura
34    // se l'utente ha inserito almeno un voto
35    if ( counter != 0 ) {
36
37      // calcola la media di tutti i voti inseriti
38      average = ( float ) total / counter; // evita il troncamento
39
40      // stampa la media con la precisione di due cifre
41      printf( "Class average is %.2f\n", average );
42    } // fine di if
43    else { //  se non sono stati inseriti voti, stampa un messaggio
44      puts( "No grades were entered" );
45    } // fine di else
46  } // fine della funzione main
```

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
```

```
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
Enter grade, -1 to end: -1
No grades were entered
```

😊 *Prevenzione di errori*

In un ciclo controllato da sentinella, ricordate esplicitamente all'utente qual è il valore della sentinella nei prompt che richiedono l'inserimento di dati.

Conversione esplicita e implicita tra tipi

Le medie non sempre hanno come valori dei numeri interi. Spesso, una media è un valore come 7,2 o 93,5 contenente una parte frazionaria.

Questi valori sono definiti numeri in virgola mobile e possono essere rappresentati dal tipo di dati float.

La variabile average viene definita di tipo float (riga 12) per catturare il risultato frazionale del nostro calcolo.

Tuttavia, il risultato del calcolo total/counter è un numero intero, poiché total e counter sono entrambe variabili intere.

La divisione di due numeri interi produce una divisione intera in cui la parte frazionale del calcolo è troncata (cioè persa).

Poiché prima viene eseguito il calcolo, la parte frazionale è perduta prima che il risultato sia assegnato ad average.

Per realizzare un calcolo in virgola mobile con valori interi, dobbiamo creare valori temporanei che sono numeri in virgola mobile.

Per fare ciò, il C mette a disposizione l'operatore **cast** unario.

La riga 38

```
average = ( float ) total / counter;
```

include l'operatore cast (float), che crea una copia temporanea in virgola mobile del suo operando, total.

Il valore memorizzato in total è ancora un numero intero.

Un simile uso dell'operatore cast è chiamato **conversione esplicita**.

Il calcolo consiste adesso in un valore in virgola mobile (la versione float temporanea di total) diviso per il valore unsigned int memorizzato in counter.

Il C calcola solo le espressioni aritmetiche in cui i tipi di dati degli operandi sono identici. Per assicurarsi che gli operandi siano dello stesso tipo, il compilatore esegue su alcuni di

essi un'operazione chiamata conversione implicita.

Ad esempio, in un'espressione contenente i tipi di dati unsigned int e float, vengono create copie degli operandi unsigned int e convertite in float.

Nel nostro esempio, dopo che viene creata una copia di counter e convertita in float, si esegue il calcolo e il risultato della divisione in virgola mobile è assegnato ad average.

Il C fornisce un insieme di regole per la conversione di operandi di tipi differenti.

Gli operatori cast sono disponibili per la maggior parte dei tipi di dati; essi vengono formati ponendo tra parentesi il nome di un tipo.

Ogni operatore cast è un operatore unario, cioè un operatore che si applica a un solo operando.

Il C supporta pure versioni unarie degli operatori più (+) e meno (–), così potete scrivere espressioni come –7 o +5.

Gli operatori cast sono associativi da destra a sinistra e hanno la stessa precedenza di altri operatori unari come + unario e – unario.

Questa precedenza è di un livello più alta di quella degli operatori moltiplicativi *, / e %.

Formattare numeri in virgola mobile

lo specificatore di conversione %.2f di printf (riga 41) viene usato per stampare il valore di average.

La f specifica che sarà stampato un valore in virgola mobile.

Il .2 è la precisione con cui il valore sarà stampato: con 2 cifre alla destra del punto decimale.

Se è usato lo specificatore di conversione %f (senza specificare la precisione), viene usata la precisione predefinita pari a 6, esattamente come se fosse stato usato lo specificatore di conversione %.6f.

Quando i valori in virgola mobile sono stampati con una certa precisione, il valore stampato è arrotondato al numero indicato di cifre decimali.

Il valore in memoria rimane inalterato.

Quando si eseguono le istruzioni seguenti, vengono stampati i valori 3.45 e 3.4 (con il punto decimale al posto della virgola, secondo la notazione del C).

```
printf( "%.2f\n", 3.446 ); // stampa 3.45
printf( "%.1f\n", 3.446 ); // stampa 3.4
```

☹ *Errore comune di programmazione*

È sbagliato usare la precisione in una specificazione di conversione nella stringa di controllo del formato di un'istruzione scanf.

Le precisioni sono usate soltanto nelle specificazioni di conversione per printf.

Note sui numeri in virgola mobile

Benché non siano sempre "precisi al 100%", i numeri in virgola mobile hanno numerose applicazioni.

Ad esempio, quando parliamo di una "normale" temperatura corporea di 37.6 gradi, non abbiamo bisogno di essere precisi fino a un grande numero di cifre.

Quando osserviamo la temperatura su un termometro e la leggiamo come 37.6, essa può in realtà essere 37.5999473210643.

Indicare questo numero semplicemente con 37.6 va benissimo per la maggior parte delle applicazioni.

Un altro possibile caso di numeri in virgola mobile è quello che si può presentare con una divisione.

Quando dividiamo 10 per 3, il risultato è 3,3333333… con la sequenza dei 3 che si ripete all'infinito.

Il computer assegna solo una quantità fissata di spazio per contenere un tale valore, così il valore in virgola mobile memorizzato può essere soltanto un'approssimazione.

☹ *Errore comune di programmazione*

Usare numeri in virgola mobile in un modo che presume che essi siano rappresentati precisamente può portare a risultati scorretti.

I numeri in virgola mobile sono rappresentati solo approssimativamente dalla maggior parte dei computer.

😖 *Prevenzione di errori*

Non confrontare i valori in virgola mobile per l'uguaglianza.

2. Operatori di assegnazione, di incremento e di decremento

Operatori di assegnazione

Il C fornisce diversi operatori di assegnazione per abbreviare le espressioni di assegnazione. Ad esempio, l'istruzione

c = c + 3;

può essere abbreviata con l'operatore di assegnazione per l'addizione += come in

c += 3;

L'operatore += aggiunge il valore dell'espressione sulla destra dell'operatore al valore della variabile sulla sinistra dell'operatore e memorizza il risultato in tale variabile. Qualunque istruzione della forma

variabile **=** espressione **operatore** variabile;

dove l'operatore è uno degli operatori binari +, –, *, / oppure %, può essere scritta nella forma

variabile **operatore=** espressione;

Così l'assegnazione c += 3 aggiunge 3 a c.

La tabella successiva mostra gli operatori di assegnazione per l'aritmetica, alcune espressioni di esempio che usano questi operatori e la loro spiegazione.

Ponete:

int c = 3, d = 5, e = 4, f = 6, g = 12;

Operatore di assegnazione	Esempio di espressione	Spiegazione	Assegna
+=	c += 7	c = c + 7	10 a c
–=	d -= 4	d = d - 4	1 a d
*=	e *= 5	e = e * 5	20 a e
/=	f /= 3	f = f / 3	2 a f
%=	g %= 9	g = g % 9	3 a g

Operatori di incremento e di decremento

Il C fornisce anche l'operatore di incremento unario ++, e l'operatore di decremento unario —.

Se una variabile va incrementata di 1, si può usare l'operatore di incremento ++ piuttosto che le espressioni c = c + 1 o c += 1.

Se gli operatori di incremento o di decremento sono posti prima di una variabile (cioè sono prefissi), sono definiti rispettivamente operatori di preincremento o di predecremento.

Se gli operatori di incremento o di decremento sono posti dopo una variabile (sono cioè postfissi), sono definiti rispettivamente operatori di postincremento o di postdecremento.

Preincrementare (o predecrementare) una variabile fa sì che essa venga prima incrementata (o decrementata) di 1 e che dopo il suo nuovo valore venga usato nell'espressione in cui appare.

Postincrementare (o postdecrementare) la variabile fa sì che sia usato il valore corrente della variabile nell'espressione in cui appare e che solo dopo il suo valore venga incrementato (o decrementato) di 1.

Operatore di assegnazione	Esempio di espressione	Spiegazione
++	++a	Incrementa a di 1, quindi usa il nuovo valore di a nell'espressione in cui si trova a.
++	a++	Usa il valore corrente di a nell'espressione in cui si trova a, quindi incrementa a di 1.
—	—b	Decrementa b di 1, quindi usa il nuovo valore di b nell'espressione in cui si trova b.
—	b—	Usa il valore corrente di b nell'espressione in cui si trova b, quindi decrementa b di 1.

Il programma successivo illustra la differenza tra le versioni di preincremento e di postincremento dell'operatore ++.

Postincrementare la variabile c fa sì che essa sia incrementata dopo essere stata usata nell'istruzione printf.

Preincrementare la variabile c fa sì che essa sia incrementata prima di essere usata nell'istruzione printf.

Il programma mostra il valore di c prima e dopo l'uso dell'operatore ++.

L'operatore di decremento (—) funziona in un modo simile.

```
01  // 10_2.c
02  // Preincremento e postincremento.
03  #include <stdio.h>
04
05  // la funzione main inizia l'esecuzione del programma
06  int main( void )
```

```
07 {
08    int c; // definizione della variabile
09
10    // illustrazione del postincremento
11    c = 5; // assegna 5 a c
12    printf( "%d\n", c ); // stampa 5
13    printf( "%d\n", c++ ); // stampa 5 e poi postincrementa
14    printf( "%d\n\n", c ); // stampa 6
15
16    // illustrazione del preincremento
17    c = 5; // assegna 5 a c
18    printf( "%d\n", c ); // stampa 5
19    printf( "%d\n", ++c ); // preincrementa e poi stampa 6
20    printf( "%d\n", c ); // stampa 6
21 } // fine della funzione main
```

```
5
5
6
5
6
6
```

😊 *Buona pratica di programmazione*

Gli operatori unari devono essere posti direttamente vicino ai loro operandi senza spazi intermedi.

Le tre istruzioni di assegnazione:

passes = passes + 1;

failures = failures + 1;

student = student + 1;

si possono scrivere più concisamente con gli operatori di assegnazione come

passes += 1;

failures += 1;

student += 1;

con operatori di preincremento come

++passes;

++failures;

++student;

o con operatori di postincremento come

passes++;

failures++;

student++;

È importante notare qui che quando si incrementa o si decrementa una variabile in un'istruzione in cui compare solo la stessa variabile, le forme di preincremento e postincremento hanno lo stesso effetto.

È solo quando una variabile appare nel contesto di un'espressione più ampia che il preincremento e il postincremento hanno effetti differenti (e lo stesso vale per il predecremento e il postdecremento).

Riguardo alle espressioni che abbiamo studiato finora, soltanto il semplice nome di una

variabile può essere usato come operando di un operatore di incremento o di decremento.

☹ *Errore comune di programmazione*

Tentare di usare l'operatore di incremento o di decremento con un'espressione diversa dal semplice nome di una variabile è un errore di sintassi: ad esempio, scrivere

++(x + 1)

☺ *Prevenzione di errori*

Il C generalmente non specifica l'ordine in cui saranno calcolati gli operandi di un operatore. Quindi dovete usare gli operatori di incremento o di decremento solamente in istruzioni nelle quali viene incrementata o decrementata una sola variabile.

La tabella successiva elenca 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.

La seconda colonna indica l'associatività degli operatori a ogni livello di precedenza.

Notate che l'operatore condizionale (?:), gli operatori unari di incremento (++) e di decremento (—), gli operatori unari più (+), meno (–), cast e gli operatori di assegnazione =, +=, –=, *=, /= e %= sono associativi da destra a sinistra.

La terza colonna denomina i vari gruppi di operatori.

Tutti gli altri operatori nella tabella sono associativi da sinistra a destra.

Operatori	Associatività	Tipo
++ (postfisso) — (postfisso)	da destra a sinistra	postfisso
+ – (tipo) ++ (prefisso) — (prefisso)	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 destra a sinistra	condizionale
= += –= *= /= %=	da destra a sinistra	di assegnazione

3. Overflow aritmetico - Programmazione sicura in C

Considerate un programma di addizione che calcola la somma di due valori int

con l'istruzione:

sum = integer1 + integer2; // assegna il totale a sum

Persino questa semplice istruzione ha un problema potenziale: addizionare numeri interi potrebbe produrre un valore troppo grande da memorizzare in una variabile int.

Ciò è noto come overflow aritmetico e può causare un comportamento indefinito, con la possibilità che il sistema venga esposto ad attacchi.

I valori massimi e minimi specifici della piattaforma che si possono memorizzare in una variabile int sono rappresentati rispettivamente dalle costanti INT_MAX e INT_MIN, che sono definite nel file di intestazione <limits.h>.

Vi sono costanti simili per gli altri tipi interi.

Potete vedere i valori relativi alla vostra piattaforma per queste costanti aprendo il file di intestazione <limits.h> in un editor di testo.

Viene considerata una buona pratica assicurarsi che, prima di eseguire calcoli aritmetici, questi non daranno luogo a overflow.

Il codice per fare questa verifica è mostrato sul sito web del CERT www.securecoding.cert.org.

Cercate proprio la linea guida "INT32-C".

Il codice utilizza gli operatori && (AND logico) e || (OR logico).

Nel codice sviluppato a livello industriale dovreste eseguire dei controlli come questi per tutti i calcoli.

Numeri interi senza segno

Abbiamo usato variabili dichiarate come unsigned int perché usate per contare solo valori non-negativi.

In generale, i contatori che memorizzano soltanto valori non-negativi devono essere dichiarati con unsigned prima del tipo intero.

Variabili di tipo unsigned possono rappresentare valori da 0 ad approssimativamente due volte l'intervallo positivo dei corrispondenti numeri di tipo intero con un segno.

Potete determinare il massimo valore unsigned int per la vostra piattaforma attraverso la costante UINT_MAX definita in <limits.h>.

Il programma della media di una classe avrebbe potuto dichiarare come unsigned int le variabili grade, total e average. I voti sono normalmente valori da 0 a 100, così total e average devono essere tutti e due maggiori o uguali a 0.

Abbiamo dichiarato quelle variabili come int perché non possiamo controllare che cosa in realtà inserisce l'utente, il quale potrebbe inserire valori negativi.

Peggio ancora, l'utente potrebbe inserire un valore che non è neppure un numero. Mostreremo come affrontare tali input in seguito.

Talvolta i cicli controllati da sentinella utilizzano valori non validi per terminare un ciclo. Ad esempio, se un programma della media voti della classe termina il ciclo quando l'utente inserisce la sentinella –1 (un voto non valido), sarebbe improprio dichiarare la variabile grade come un unsigned int.

Esiste un indicatore della fine del file (EOF) spesso usato per terminare i cicli controllati da sentinella. E' un numero negativo.

scanf_s e printf_s

L'Annex K dello standard C11 introduce versioni più sicure di printf e scanf chiamate printf_s e scanf_s.

L'Annex K è designato come opzionale, per cui non sarà implementato da tutti i fornitori del linguaggio C.

Microsoft ha implementato le sue versioni di printf_s e scanf_s prima della pubblicazione dello standard C11, e il suo compilatore ha cominciato immediatamente a far emettere dei warning (avvertimenti) per ogni chiamata di scanf.

I warning dicono che scanf è deprecata – non deve essere più utilizzata – e che voi dovreste prendere in considerazione invece l'uso di scanf_s.

Molte organizzazioni hanno standard di codifica che richiedono che il codice venga compilato senza messaggi di warning.

Vi sono due modi per eliminare i warning per scanf in Visual C++:

potete usare scanf_s invece di scanf, oppure potete disattivare questi warning.

Riguardo alle istruzioni di input che abbiamo usato finora, gli utenti di Visual C++ possono semplicemente sostituire scanf con scanf_s.

Potete disattivare i messaggi di warning in Visual C++ come segue:

1. Digitate Alt F7 per far apparire la finestra Property Pages per il vostro progetto.

2. Nella colonna sinistra, espandete Configuration Properties>C/C++ e selezionate Preprocessor.

3. Nella colonna destra, alla fine del valore per Preprocessor Definitions, inserite

;_CRT_SECURE_NO_WARNINGS

4. Cliccate OK per salvare le modifiche.

Non riceverete più warning per scanf (o per qualunque altra funzione che Microsoft ha deprecato per ragioni simili).

Per codice a livello industriale è sconsigliato disattivare i warning.

Bibliografia

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