Introduzione alla programmazione in C

Un esempio di programma in C

```
1. // fig02_01.c
2. // A first program in C.
3. #include <stdio.h>
4.
5. // function main begins program execution
6. int main(void) {
7.    printf("Welcome to C!\n");
8. } // end function main
```

Output: Welcome to C!

Un esempio di programma in C

```
// A first program in C.
```

- I commenti iniziano con **//**
- Necessario per la **documentazione** di un programma e migliorare la leggibilità
- I commenti non eseguono nessuna azione
- Aiutano le altre persone a leggere e capire il vostro programma
- Si usa **/*...*/** per un commento multi-linea
  - Tutto ciò tra **/*** and ***/** è un commento

Un esempio di programma in C

```
#include <stdio.h>
```

- **Indicazione per il processor di C**
- Il preprocessore si occupa delle righe che iniziano con # prima della compilazione
- Questa istruzione include nel programma il contenuto dello standard input/output header (`<stdio.h>`)
  - Contiene informazioni (funzioni e variabili) che il compilatore utilizza l'I/O standard (e.g., `printf`)

Un esempio di programma in C

Righe vuote (“blank lines”) e ”white space”

- Le righe vuote, gli spazi e i caratteri di tabulazione rendono i programmi più facili da leggere
- Insieme, questi elementi sono noti come spazi bianchi (white space)
- Sono generalmente ignorati dal compilatore
- Importante "indentare" il codice per migliorare la leggibilità

Un esempio di programma in C

```
// function main begins program execution
int main(void){
```

- Il main inizia l'esecuzione di ogni programma C
- Le parentesi indicano che il main è una **funzione**
- Tutti i programmi C consistono di funzioni, incluso il main
- Utile precedere ogni funzione con un commento per descriverla
- Le funzioni "ritornano" delle informazioni
- La keyword `int` indica che il main ritorna un numero intero

Un esempio di programma in C

```
// function main begins program execution
int main(void){
```

- Le funzioni possono anche ricevere informazioni
  - `void` in questo caso indica che non riceve nulla
- Il corpo (body) di una funzione inizia e termina con delle parentesi graffe
- Le parentesi formano un blocco (function block)

Un esempio di programma in C

```
printf("Welcome to C!\n");
```

Output: Welcome to C!

- Esempio di **output**
- "f" in `printf` indica "formattato"
- La funzione stampa a schermo la stringa tra virgolette
  - Una stringa è detta anche stringa di caratteri, messaggio o literal
- L'intera riga è detta **statement**
- Ogni statement termina con un `;` (**statement terminator)**
- I char vengono stampati così come appaiono
  - Notare che il carattere `\n` non viene stampato

Un esempio di programma in C

Sequenze di Escape

- In una stringa, il backslash (`\`) is è un **escape character**
- Il Compiler combina un backslash con il carattere successive per formare una **escape sequence**
- `\n` indica **newline (“a capo”)**
- Quando `printf` incontra un newline in una stringa, posiziona il cursore di output all’inizio della prossima riga

Un esempio di programma in C

Escape Sequence	Description
`\n`	Moves the cursor to the beginning of the next line.
`\t`	Moves the cursor to the next horizontal tab stop.
`\a`	Produces a sound or visible alert without changing the current cursor position.
`\\`	Because the backslash has special meaning in a string, **** is required to insert a backslash character in a string.
`\"`	Because strings are enclosed in double quotes, **\"** is required to insert a double-quote character in a string.

Fonte: Deitel & Deitel

Un esempio di programma in C

Linker ed Executables

- Quando si compila una dichiarazione `printf`, il compilatore riserva semplicemente spazio nel programma oggetto per una "chiamata" alla funzione
- Il compilatore non sa dove si trovano le funzioni di libreria–lo sa il **linker**
- Quando viene eseguito, il linker individua le funzioni di libreria e inserisce le chiamate corrette a queste funzioni nel programma oggetto
- Ora il programma oggetto è completo e pronto per l'esecuzione
- Il programma collegato è chiamato **eseguibile**

Un esempio di programma in C

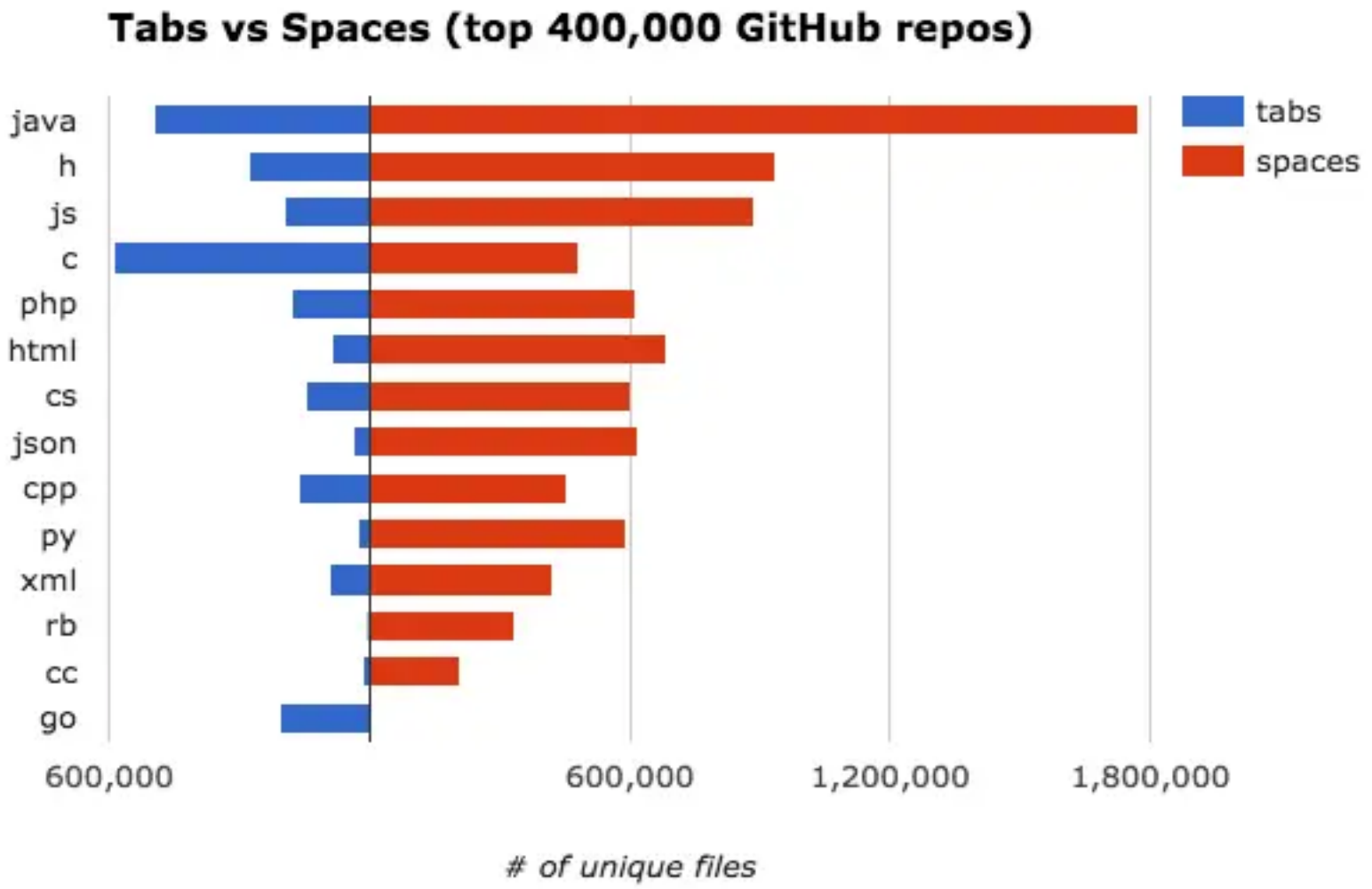
Convenzioni di indentazione

- Indentare tutto il corpo di ciascuna funzione di un livello all'interno delle parentesi graffe che definiscono il corpo della funzione
- Questo enfatizza la struttura funzionale di un programma e ne facilita la lettura
- Stabilire una convenzione di indentazione e applicarla uniformemente
- Le guide di stile raccomandano spesso di usare **spazi** anziché **tabulazioni**
- Gli ambienti di programmazione (spesso) indentano in automatico

```
1  #include <stdio.h>
2  int main()
3  {
4      int i=1,j;
5      while (i <= 5)
6      {
7          j=1;
8          while (j <= i )
9          {
10             printf("%d ",j);
11             j++;
12         }
13         printf("\n");
14         i++;
15     }
16     return 0;
17 }
```

Fonte: it.emcelettronica.com

Tabs vs Spaces



Fonte: Hoffa/Medium

# of unique files

Fonte: Hoffa/Medium

Un esempio di programma in C

```
1. // fig02_02.c
2. // Printing on one line with two printf
   statements.
3. #include <stdio.h>
4.
5. // function main begins program execution
6. int main(void) {
7.    printf("Welcome ");
8.    printf("to C!\n");
9. } // end function main
```

- **Output**: Welcome to C!

Un esempio di programma in C

```
printf("Welcome ");
printf("to C!\n");
```

Usare `printf` multipli

- Il nuovo codice utilizza due istruzioni per ottenere lo stesso output
- Funziona perché ogni `printf` riprende la stampa da dove si è fermato quello precedente.
- La riga 7 visualizza "Welcome" seguito da uno spazio (ma senza andare a capo)
- Il printf della riga 8 inizia a stampare immediatamente dopo lo spazio

Un esempio di programma in C

```
1. // fig02_03.c
2. // Printing multiple lines with a single printf.
3. #include <stdio.h>
4.
5. // function main begins program execution
6. int main(void) {
7.    printf("Welcome\nto\nC!\n");
8. } // end function main
```

- **OUTPUT:**

  Welcome
  to
  C!

Visualizzare più righe con un singolo `printf`:

- Un solo `printf` può visualizzare diverse righe
- Ogni `\n` sposta il cursore dell'output all'inizio della riga successiva

Istruzioni per stampare

- **`printf`**:
  - Utilizzata per stampare output formattato su standard output (generalmente il terminale)
  - Consente di specificare il formato dell'output tramite specificatori come `%d` (interi), `%f` (numeri in virgola mobile), `%s` (stringhe), ecc
  - Permette di controllare la larghezza del campo, la precisione e l'allineamento dell'output
- **`puts`**:
  - Utilizzata per stampare una stringa su standard output seguita da un carattere di nuova linea (`\n`)
    - Va a capo automaticamente
  - Non supporta formattazione avanzata; stampa solo stringhe come sono

Quiz

Quiz

Quiz

Un altro esempio: sommare due numeri interi

```
1. // fig02_04.c
2. // Addition program.
3. #include <stdio.h>
4. // function main begins program execution
5. int main(void) {
6.    int integer1 = 0; // will hold first number user enters
7.    int integer2 = 0; // will hold second number user enters
8.
9.    printf("Enter first integer: "); // prompt
10.   scanf("%d", &integer1); // read an integer
11.
```

Un altro esempio: sommare due numeri interi

```
12.
13. printf("Enter second integer: "); // prompt
14.  scanf("%d", &integer2); // read an integer
15.
16.  int sum = 0; // variable in which sum will be stored
17.  sum = integer1 + integer2; // assign total to sum
18.
19.  printf("Sum is %d\n", sum); // print sum
20. } // end function main
```

Un altro esempio: sommare due numeri interi

- La funzione `scanf` della libreria standard ottiene informazioni dall'utente tramite tastiera
- Il programma somma i due interi e stampa il risultato
- `scanf` utilizza specificatori di formato per leggere i dati dal flusso di input e memorizzarli nelle variabili appropriate
- Il risultato della somma viene visualizzato utilizzando la funzione `printf`

Un altro esempio: sommare due numeri interi

- OUTPUT:

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Un altro esempio: sommare due numeri interi

```
 int integer1 = 0; // will hold first
number user enters
 int integer2 = 0; // will hold second
number user enters
```

- Queste righe sono delle **definizioni**
- I nomi `integer1` e `integer2` indicano della **variabili**—spazi in memoria dove il programma salva dei valori
- `integer1` e `integer2` sono di tipo **`int`**—possono rappresentare soltanto dei numeri interi
- Queste righe inizializzano a 0 le due variabili

Un altro esempio: sommare due numeri interi

Bisogna definire le variabili prima del loro utilizzo!

- Tutte le variabili devono essere definite con un nome e un tipo prima di poter essere utilizzate in un programma
- È possibile posizionare ciascuna definizione di variabile in qualsiasi punto di main, purché prima del primo utilizzo della variabile nel codice
- In generale, è consigliabile definire le variabili vicino al loro primo utilizzo

Un altro esempio: sommare due numeri interi

Identificatori e sensibilità alle maiuscole:

- Un nome di variabile può essere qualsiasi identificatore valido
- Ogni identificatore può essere composto da lettere, cifre e underscore (_), ma non può iniziare con una cifra
- C è case sensitive, quindi `a1` e `A1` sono identificatori diversi
- Un nome di variabile dovrebbe iniziare con una lettera minuscola
- Scegliere nomi di variabili significativi aiuta a rendere un programma auto-documentante
- I nomi di variabili composti da più parole possono migliorare la leggibilità dei programmi:
  - separando le parole con underscore, come in `total_commissions`, oppure
  - unendo le parole e iniziando ogni parola successiva con una maiuscola, come in `totalCommissions`

Un altro esempio: sommare due numeri interi

```
printf("Enter first integer: "); // prompt
```

Messaggi di prompting:

- Prima di leggere l'input, il programma stampa a schermo `"Enter first integer: "` utilizzando `printf`
- Questo messaggio si chiama **prompt**
  - Chiede all'utente di compiere un'azione specifica

Un altro esempio: sommare due numeri interi

```
scanf("%d", &integer1); // read an integer
```

La funzione scanf e gli input formattati:

- Il programma utilizza `scanf` per ottenere un valore dall'utente.
- Legge dall'input standard, di solito la tastiera
- La stringa di controllo del formato `"%d"` indica il tipo di dati che l'utente deve inserire (un intero)
- Il secondo argomento inizia con un ampersand (`&`) seguito dal nome della variabile
- Indica a scanf la posizione (o indirizzo) in memoria della variabile
- scanf memorizza il valore inserito dall'utente in quella posizione di memoria

Un altro esempio: sommare due numeri interi

```
printf("Enter second integer: "); // prompt
scanf("%d", &integer2); // read an integer


int sum = 0; // variable in which sum will be
stored
sum = integer1 + integer2; // assign total to sum
```

Richiesta e inserimento del secondo intero:

- Il programma chiede all'utente di inserire il secondo intero
- Salva un valore per la variabile `integer2`

Definizione della variabile `sum`:

- Il programma definisce la variabile `int sum` e la inizializza a 0 prima di usarla

Un altro esempio: sommare due numeri interi

Istruzione di assegnazione:

- L'istruzione di assegnazione `sum = integer1 + integer2;` calcola la somma di `integer1` e `integer2`, quindi assegna il risultato alla variabile sum utilizzando l'operatore di assegnazione (=)
- Si legge come: “sum riceve il valore dell'espressione integer1 + integer2.»
- La maggior parte dei calcoli viene eseguita nelle assegnazioni

Operatori binari:

- Gli operatori = e + sono operatori binari, ognuno dei quali ha due operandi
- Utile inserire spazi su entrambi i lati di un operatore binario per farlo risaltare e rendere il programma più leggibile

Un altro esempio: sommare due numeri interi

```
printf("Sum is %d\n", sum); // print sum
```

Stampa con una stringa di controllo del formato:

- La stringa di controllo del formato `"Sum is %d\n"` contiene alcuni caratteri letterali da visualizzare ("Sum is ") e la specifica di conversione `%d`, che è un segnaposto per un intero
- Il valore di sum è quello da inserire al posto di `%d`

Combinare una definizione di variabile e un'istruzione di assegnazione:

- È possibile inizializzare una variabile nella sua definizione
- Ad esempio:
  - `int sum = integer1 + integer2; // assegna il totale a sum`

Un altro esempio: sommare due numeri interi

Calcoli negli statement `printf`:

- Non avremmo bisogno della variabile sum perchè possiamo eseguire dei calcoli direttamente nella printf
- Possiamo quindi sostituire con:
  - `printf("Sum is %d\n", integer1 + integer2);`

Intervallo

Fonte: PlaygroundAI

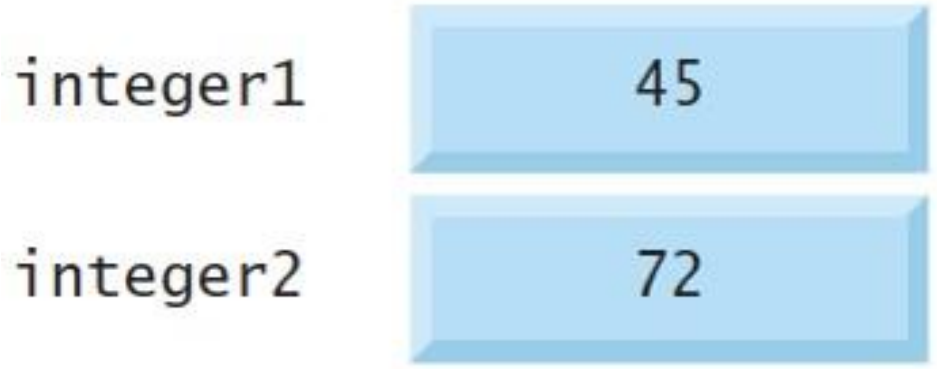
Memoria

- Ogni variabile ha un **nome, tipo, valore e posizione** nella memoria del computer
- Dopo aver caricato 45 nella variabile `integer1`

- Quando un valore viene posizionato in memoria, sostituisce il valore precedente che viene perso

Memoria

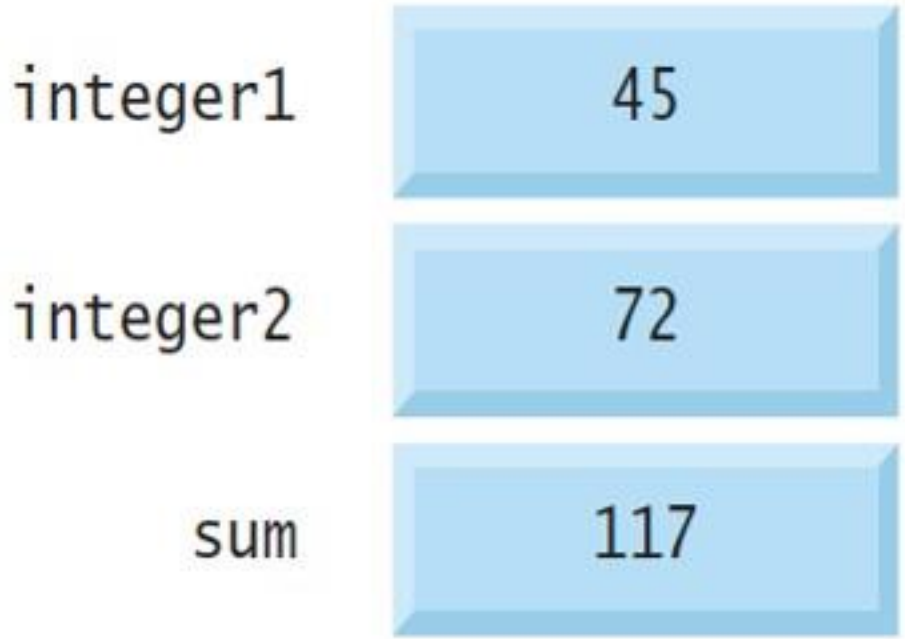
- Salviamo 72 nella variabile `integer2`



- Queste posizioni NON sono necessariamente adiacenti nella memoria

Memoria

- Dopo aver calcolato la somma di `integer1` e `integer2`



- Il valore 117 viene salvato nella posizione in memoria indicate dalla variabile `sum`

Aritmetica in C

- Operatori aritmetici binari

C operation	**Arithmetic operator**	**Algebraic expression**	**C expression**
Addizione	+	$f + 7$	f + 7
Sottrazione	–	$p - c$	`p - c`
Moltiplicazione	*	bm	`b * m`
Divisione	/	x/y or $\frac{x}{y}$	`x / y`
Resto	%	$r \bmod s$	`r % s`

Aritmetica in C

Divisione tra interi e Resto

- La divisione tra interi restituisce un numero intero
  - `7 / 4` ritorna 1
  - `17 / 5` ritorna 3
- L'operatore **resto %** restituisce il resto dopo la divisione tra interi
  - `7 % 4` ritorna 3
  - `17 % 5` ritorna 2
- Dividere per zero è un'operazione **undefined:**
  - Generalmente un errore fatale
  - Gli errori non fatali invece permettono al programma di continuare ad eseguire ma spesso con risultati sbagliati

Aritmetica in C

Parentesi per raggruppare espressioni

- Le parentesi in C vengono usate per raggruppare le espressioni come in algebra

Regole di precedenza

- Generalmente come in algebra
  - Le espressioni tra parentesi vengono eseguite prima
    - Nelle parentesi **annidate**, gli operatori più interni vengono applicati prima
  - *, / e % si applicano da sinistra e destra (left-to-right)
  - + and − sono eseguiti successivamente left-to-right
  - L'operatore di assegnazione (=) viene eseguito per ultimo

Aritmetica in C

Algebra: $m = \frac{a+b+c+d+e}{5}$

C:

```
m=(a+b+c+d+e)/5;
```

- Rispetto all'algebra, in questo caso in C servono le parentesi

Algebra: $y = mx + b$

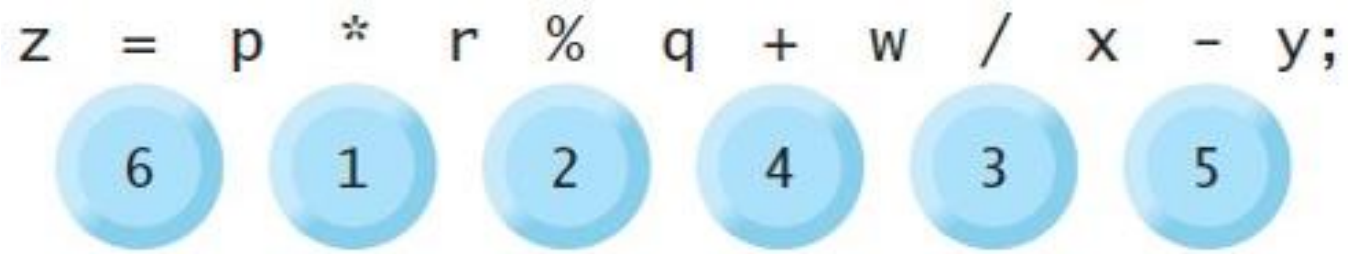
C:

```
y=m*x+b;
```

Aritmetica in C

Algebra: $z = pr \bmod q + w/x - y$

C:



Aritmetica in C

- Polinomio di secondo grado in C

Aritmetica in C

Siano a = 2, b = 3, c = 7 e x = 5

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72` *(Last operation—place 72 in y)*

Aritmetica in C

Utilizzare le parentesi per migliorare la leggibilità

- **Parentesi ridondanti** possono rendere un'espressione più chiara
  - `y = (a * x * x) + (b * x) + c;`

Operatori di uguaglianza e relazionali

- Le istruzioni eseguibili eseguono azioni come calcoli, input e output, o prendono **decisioni**
- Una **condizione** è un'espressione che può essere **vera** o **falsa**
- l'istruzione `if` prende una decisione basata sul valore della condizione
  - Se la condizione è vera, l'istruzione nel corpo dell'`if` viene eseguita
  - Altrimenti, non viene eseguita
- Le condizioni sono formate utilizzando gli operatori di uguaglianza e relazionali

Operatori di uguaglianza e relazionali

Relational operators

Algebraic equality or relational operator	C equality or relational operator	Sample C condition	Meaning of C condition
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Equality operators

Algebraic equality or relational operator	C equality or relational operator	Sample C condition	Meaning of C condition
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Operatori di uguaglianza e relazionali

Confondere uguaglianza == e assegnazione =

- Errore di programmazione comune
- Leggere l'operatore di uguaglianza come “doppio uguale” e l'operatore di assegnazione come “assegna” o “riceve il valore di”
- Confondere questi operatori può causare **errori logici** difficili da trovare anziché errori di compilazione

Operatori di uguaglianza e relazionali

```
1.  // fig02_05.c
2.  // Using if statements, relational
3.  // operators, and equality operators.
4.  #include <stdio.h>
5.
6.  // function main begins program execution
7.  int main(void) {
8.     printf("Enter two integers, and I will tell you\n");
9.     printf("the relationships they satisfy: ");
10.
11.    int number1 = 0; // first number to be read from user
12.    int number2 = 0; // second number to be read from user
13.
14.    scanf("%d %d", &number1, &number2); // read two integers
15.
```

Operatori di uguaglianza e relazionali

```
16.  if (number1 == number2){
17.        printf("%d is equal to %d\n", number1, number2);
18. }
19.     // end if
20.
21.    if (number1 != number2) {
22.       printf("%d is not equal to %d\n", number1, number2);
23.    } // end if
24.
25.    if (number1 < number2) {
26.       printf("%d is less than %d\n", number1, number2);
27.    } // end if
28.
```

Operatori di uguaglianza e relazionali

```
28.  if (number1 > number2) {
29.        printf("%d is greater than %d\n", number1, number2);
30.    } // end if
31.
32.    if (number1 <= number2) {
33.        printf("%d is less than or equal to %d\n", number1,
    number2);
34.    } // end if
35.
36.    if (number1 >= number2) {
37.        printf("%d is greater than or equal to %d\n", number1,
    number2);
38.    } // end if
39. } // end function main
```

Operatori di uguaglianza e relazionali

- OUTPUT 1:

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

Operatori di uguaglianza e relazionali

- OUTPUT 2:

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

Operatori di uguaglianza e relazionali

- OUTPUT 3:

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Operatori di uguaglianza e relazionali

Confronto di due numeri

- Ogni istruzione `if` controlla una condizione diversa sui valori di `number1` e `number2`
- Indentando il corpo di ciascuna istruzione `if` e inserendo righe vuote sopra e sotto ogni istruzione `if` migliora la leggibilità del programma
- È possibile inserire qualsiasi numero di istruzioni all'interno del corpo di un'istruzione `if`
- Inserire un punto e virgola immediatamente a destra della parentesi destra dopo la condizione di un'istruzione `if` è un errore comune
- Il punto e virgola viene trattato come un'istruzione vuota che non esegue alcun compito

Operatori di uguaglianza e relazionali

Keyword

- Alcune parole come `int`, `if` e `void`, sono **keyword** o **parole riservate** del linguaggio di programmazione e hanno un significato particolare per il compilatore
- NON UTILIZZARLE COME IDENTIFICATORI (nomi variabili)!

Operatori di uguaglianza e relazionali

Keyword

`auto`	`do`	`goto`	`signed`	`unsigned`
`break`	`double`	`if`	`sizeof`	`void`
`case`	`else`	`int`	`static`	`volatile`
`char`	`enum`	`long`	`struct`	`while`
`const`	`extern`	`register`	`switch`	
`continue`	`float`	`return`	`typedef`	
`default`	`for`	`short`	`union`	

Keyword aggiunte nello standard C99

```
_Bool _Complex _Imaginary inline restrict
```

Keyword aggiunte nello standard C11

```
_Alignas _Alignof _Atomic _Generic _Noreturn _Static_assert
_Thread_local
```

Quiz

Quiz

Recap

- Esempi di programmi in C
  - Commenti
  - Funzioni main, printf e scanf
  - Librerie
  - Indentazione
- Memoria e variabili in C
- Aritmetica in C
- Operatori di uguaglianza e relazionali