

Laboratorio di Informatica

Linguaggio C

(Parte 1)

docente: Cataldo Musto

cataldo.musto@uniba.it

Linguaggio C

- **Perchè il linguaggio C ?**
 - C è **piccolo** (numero limitato di istruzioni)
 - Abbastanza **semplice da imparare**
 - E' **efficiente**
 - Caratterizzato da un buon livello di **portabilità**



Linguaggio C

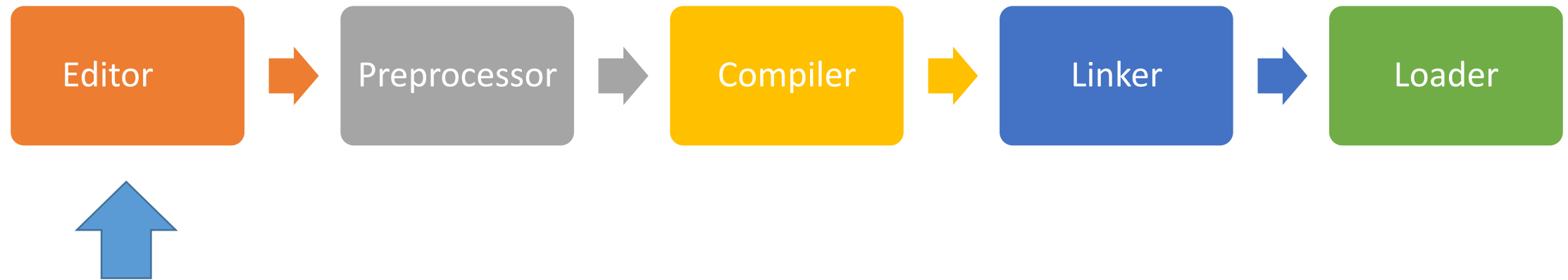
- **C**
 - Evoluzione di due precedenti linguaggi di programmazione, BCPL e B (by Ritchie)
 - Usato per scrivere i **moderni sistemi operativi**
 - Il sistema operativo UNIX è scritto in C
- **Standardizzazione**
 - Inizialmente esistevano diverse varianti del C, ma non erano compatibili tra di loro. E' stato formato un comitato per creare una **definizione "non ambigua, e machine-independent"**
 - Standard creato nel 1989, **aggiornato nel 1999 (la versione più comune, detta C-99)** ri-aggiornato nel 2011



Recap: compilazione codice sorgente

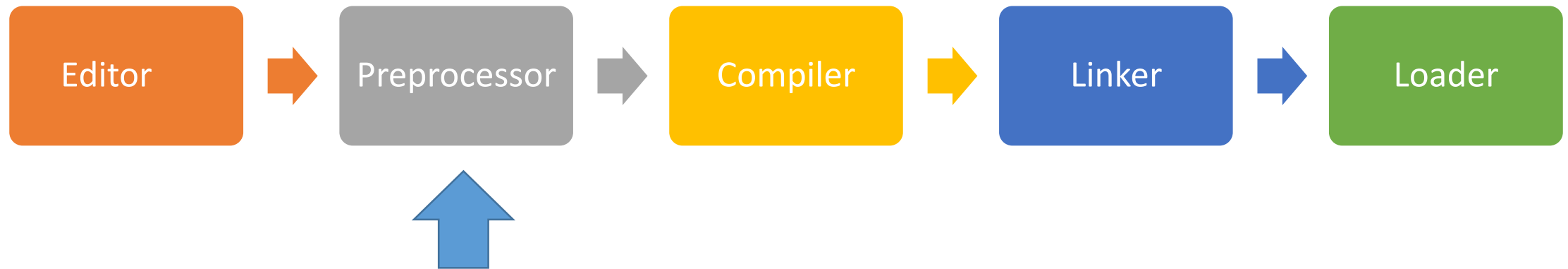


Recap: compilazione codice sorgente



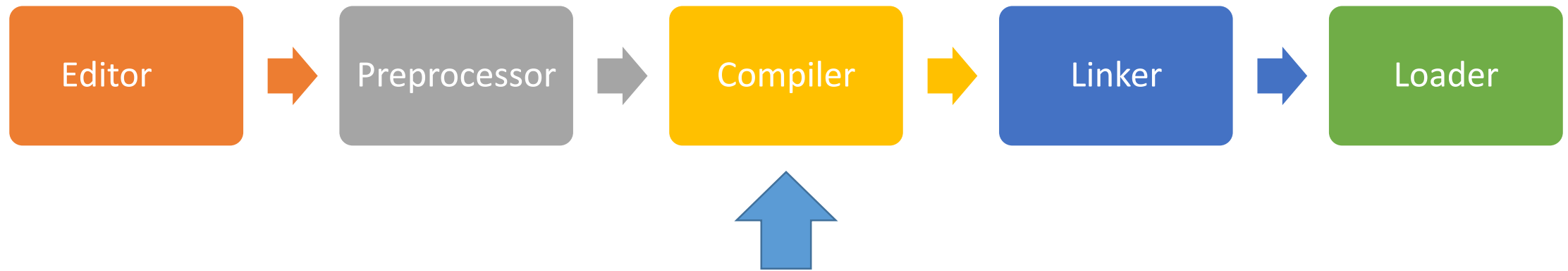
Scrittura del **codice sorgente**
(in un IDE o con un semplice file di testo)

Recap: compilazione codice sorgente



Risoluzione delle direttive, ad esempio
#define (utilizzata per definire costanti, ad esempio) e **#include** (utilizzata per includere codice scritto in librerie esterne)

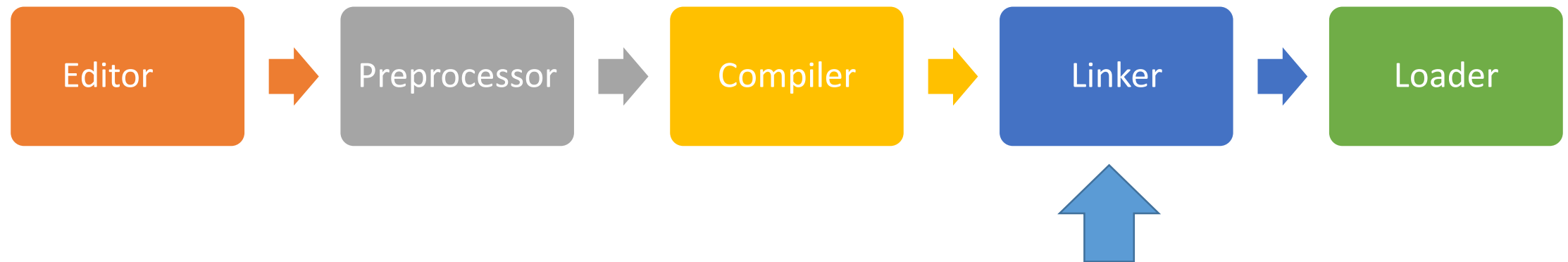
Recap: compilazione codice sorgente



Verifica la correttezza sintattica del codice sorgente e costruisce un file oggetto (con estensione **.o**) che viene salvato su disco.

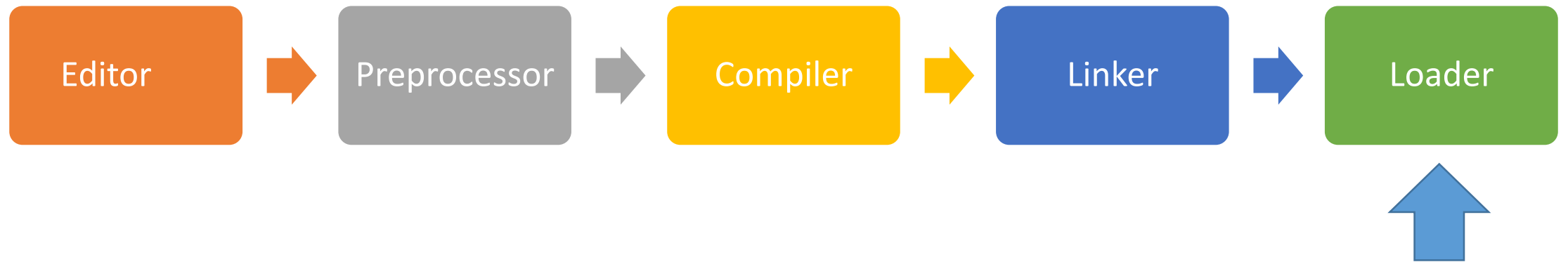
Errori logici (reversibili o irreversibili) non vengono individuati dal compilatore.

Recap: compilazione codice sorgente



Collega i **vari file oggetto** costruiti dal compilatore e unisce eventuali **librerie esterne**, al fine di generare il file eseguibile

Recap: compilazione codice sorgente




Carica in memoria e lancia
l'eseguibile compilato. Il processo è
preso in carico dalla CPU che esegue
sequenzialmente le istruzioni ed
eventualmente alloca della memoria
per creare variabili, file su disco, etc.

Linguaggio C: il primo programma

```
1      #include <stdio.h>
3      int main()
4      {
5          int integer1;
6          int integer2;
7          int sum;
8
9          printf(«Enter value: »);
10         scanf(«%d», &integer1);
11
12         printf(«Enter value: »);
13         scanf(«%d», &integer2);
14
15         sum = integer1 + integer2;
16         printf(«Sum: %d\n», sum);
17
18         return 0;
19     }
```

Linguaggio C: il primo programma


```
1  #include <stdio.h>
3  int main()
4  {
5      int integer1;
6      int integer2;
7      int sum;
8
9      printf(«Enter value: »);
10     scanf(«%d», &integer1);
11
12     printf(«Enter value: »);
13     scanf(«%d», &integer2);
14
15     sum = integer1 + integer2;
16     printf(«Sum: %d\n», sum);
17
18     return 0;
19 }
```



Riga 1: direttive al preprocessore.
Aggiunge le funzioni per gestire i flussi di input/output

Linguaggio C: il primo programma

```
1  #include <stdio.h>
3  int main()
4  {
5      int integer1;
6      int integer2;
7      int sum;
8
9      printf(«Enter value: »);
10     scanf(«%d», &integer1);
11
12     printf(«Enter value: »);
13     scanf(«%d», &integer2);
14
15     sum = integer1 + integer2;
16     printf(«Sum: %d\n», sum);
17
18     return 0;
19 }
```



Riga 1: direttive al preprocessore.
Aggiunge le funzioni per gestire i flussi di input/output

Riga 3: il main è la funzione principale

Linguaggio C: il primo programma

```
1    #include <stdio.h>
3    int main()
4    {
5        int integer1;
6        int integer2;
7        int sum;
8
9        printf(«Enter value: »);
10       scanf(«%d», &integer1);
11
12       printf(«Enter value: »);
13       scanf(«%d», &integer2);
14
15       sum = integer1 + integer2;
16       printf(«Sum: %d\n», sum);
17
18       return 0;
19   }
```

Riga 1: direttive al preprocessore.
Aggiunge le funzioni per gestire i flussi di input/output

Riga 3: il main è la funzione principale

Riga 5-7: dichiariamo variabili di tipo intero



Linguaggio C: il primo programma

```
1      #include <stdio.h>
3      int main()
4      {
5          int integer1;
6          int integer2;
7          int sum;
8
9          printf(«Enter value: »);
10         scanf(«%d», &integer1);
11
12         printf(«Enter value: »);
13         scanf(«%d», &integer2);
14
15         sum = integer1 + integer2;
16         printf(«Sum: %d\n», sum);
17
18         return 0;
19     }
```

Riga 1: direttive al preprocessore.
Aggiunge le funzioni per gestire i flussi di input/output

Riga 3: il main è la funzione principale

Riga 5-7: dichiariamo variabili di tipo intero

Riga 9-10: stampa di una stringa, lettura di un valore e memorizzazione in una variabile.



Linguaggio C: il primo programma

```
1      #include <stdio.h>
3      int main()
4      {
5          int integer1;
6          int integer2;
7          int sum;
8
9          printf(«Enter value: »);
10         scanf(«%d», &integer1);
11
12         printf(«Enter value: »);
13         scanf(«%d», &integer2);
14
15         sum = integer1 + integer2;
16         printf(«Sum: %d\n», sum);
17
18         return 0;
19     }
```

Riga 1: direttive al preprocessore.
Aggiunge le funzioni per gestire i flussi di input/output

Riga 3: il main è la funzione principale

Riga 5-7: dichiariamo variabili di tipo intero

Riga 9-10: stampa di una stringa, lettura di un valore e memorizzazione in una variabile.

Perché utilizziamo &?



Linguaggio C: il primo programma

```
1    #include <stdio.h>
3    int main()
4    {
5        int integer1;
6        int integer2;
7        int sum
8
9        printf(«Enter value: »);
10       scanf(«%d», &integer1);
11
12       printf(«Enter value: »);
13       scanf(«%d», &integer2);
14
15       sum = integer1 + integer2;
16       printf(«Sum: %d\n», sum);
17
18       return 0
19   }
```

Riga 1: direttive al preprocessore.

Aggiunge le funzioni per gestire i flussi di input/output

Riga 3: il main è la funzione principale

Riga 5-7: dichiariamo variabili di tipo intero

Riga 9-10: stampa di una stringa, lettura di un valore e memorizzazione in una variabile. **Ricordiamo che una variabile è un nome che identifica una locazione di memoria, quindi serve l'operatore & per referenziare l'indirizzo di quella variabile**



Linguaggio C: il primo programma

```
1    #include <stdio.h>
3    int main()
4    {
5        int integer1;
6        int integer2;
7        int sum;
8
9        printf(«Enter value: »);
10       scanf(«%d», &integer1);
11
12       printf(«Enter value: »);
13       scanf(«%d», &integer2);
14
15       sum = integer1 + integer2;
16       printf(«Sum: %d\n», sum);
17
18       return 0;
19   }
```

Riga 1: direttive al preprocessore.

Aggiunge le funzioni per gestire i flussi di input/output

Riga 3: il main è la funzione principale

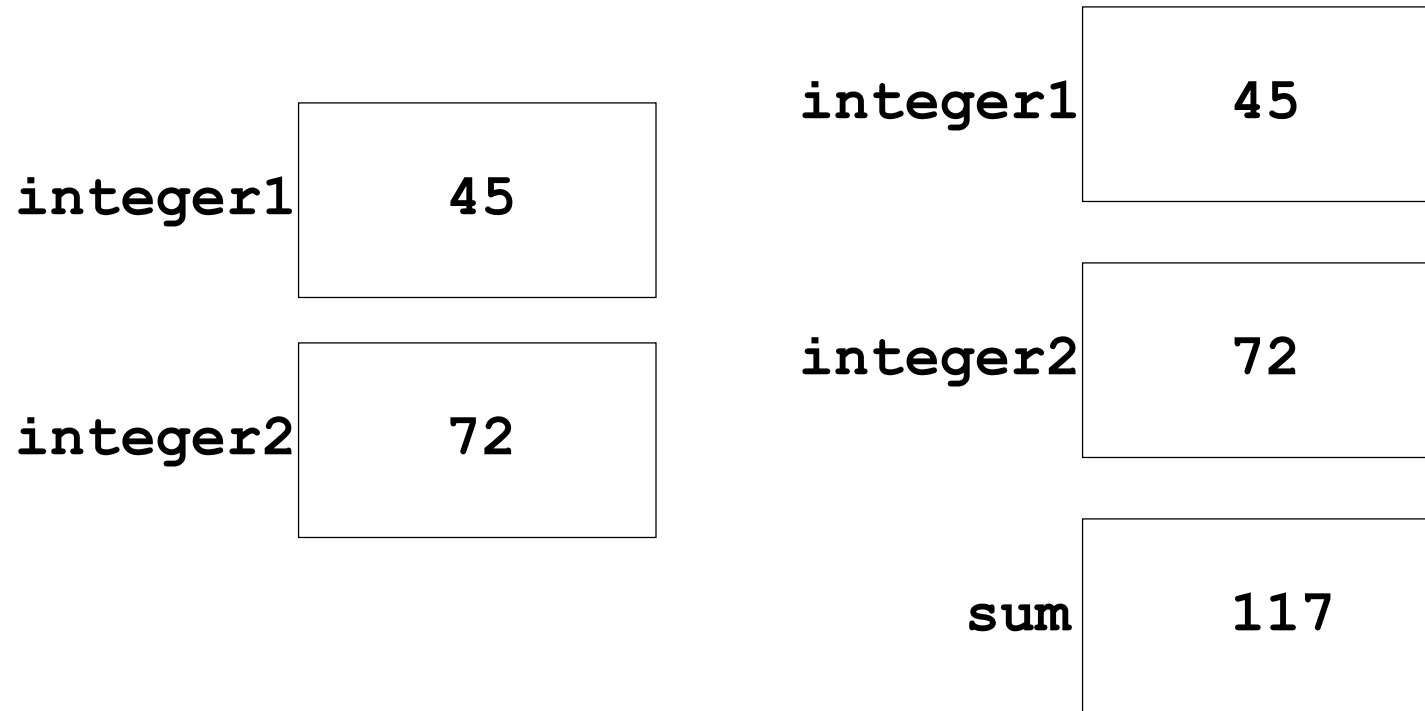
Riga 5-7: dichiariamo variabili di tipo intero

Riga 9-10: stampa di una stringa, lettura di un valore e memorizzazione in una variabile. **Ricordiamo che una variabile è un nome che identifica una locazione di memoria, quindi serve l'operatore & per referenziare l'indirizzo di quella variabile**

Riga 15-17: somma aritmetica, assegnazione di un valore a una nuova variabile e stampa del valore.

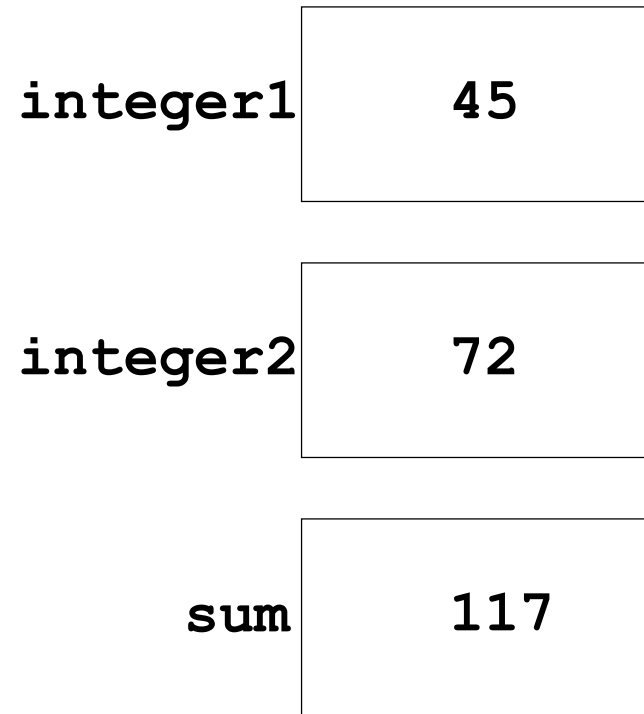
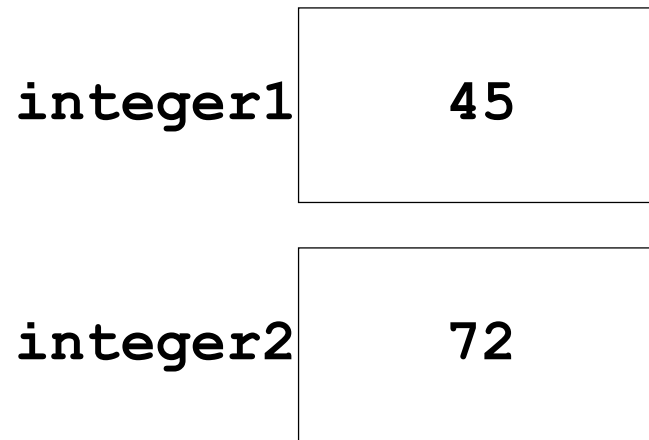


Linguaggio C: cenni sulla memoria



Ogni variabile è identificata attraverso tre elementi. **Quali?**

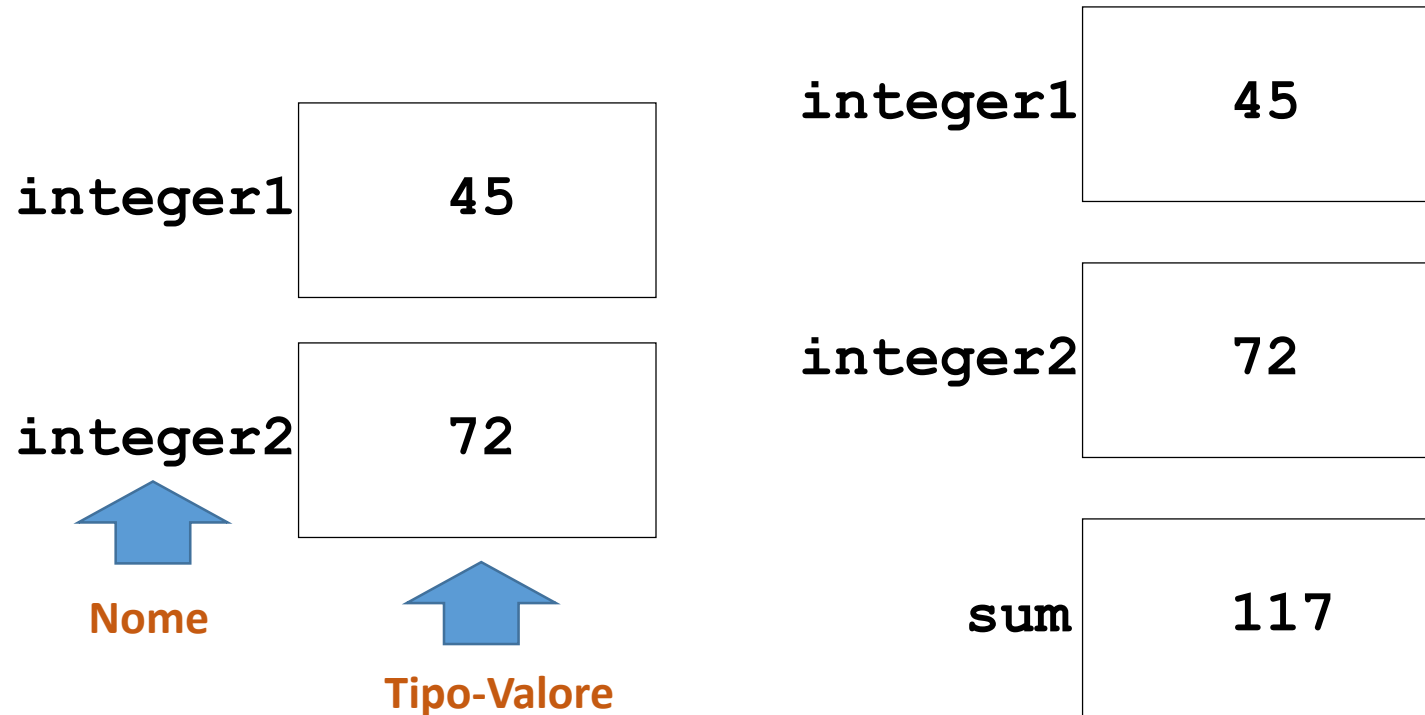
Linguaggio C: cenni sulla memoria



Ogni variabile è identificata attraverso tre elementi. **Quali?**

- **Nome**
- **Tipo**
- **Valore**

Linguaggio C: cenni sulla memoria

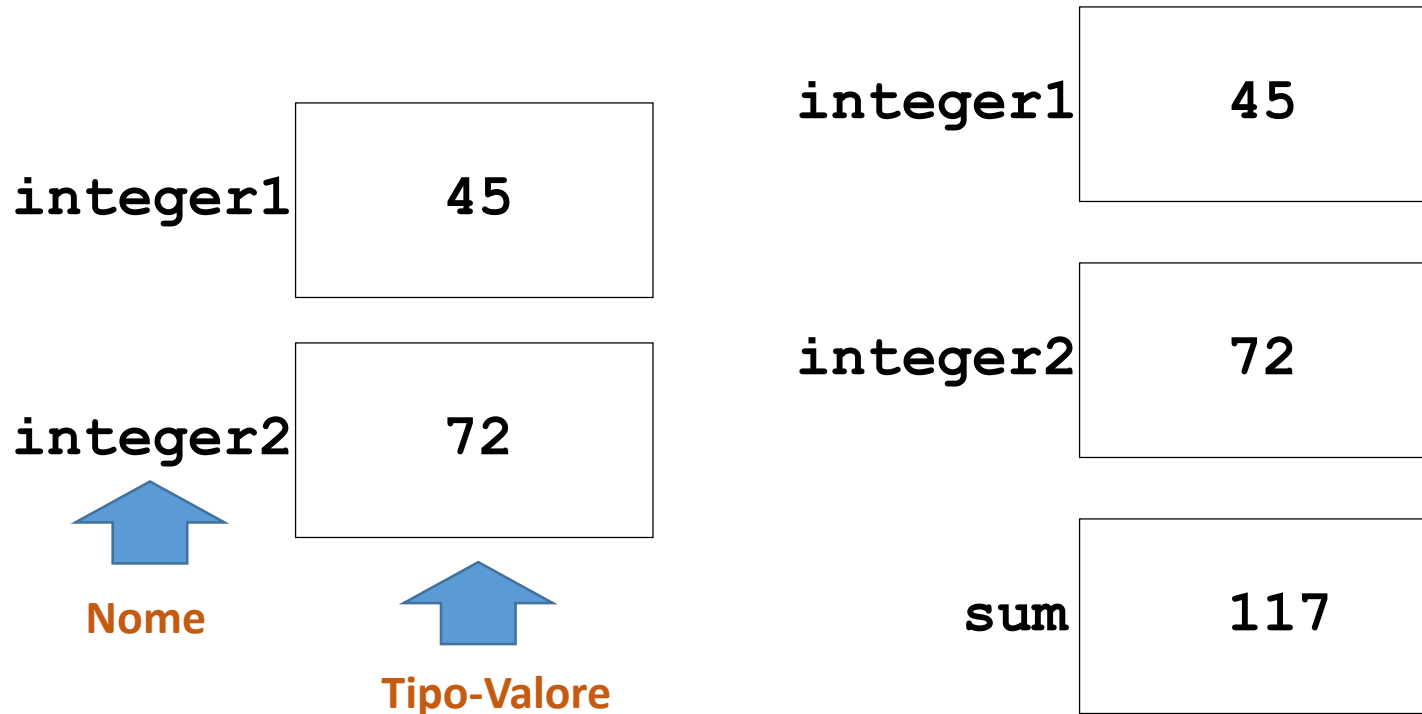


Ogni variabile è identificata attraverso tre elementi. **Quali?**

- **Nome**
- **Tipo**
- **Valore**

Quando dichiariamo una variabile è importante inicializzarla. **Perché?**

Linguaggio C: cenni sulla memoria



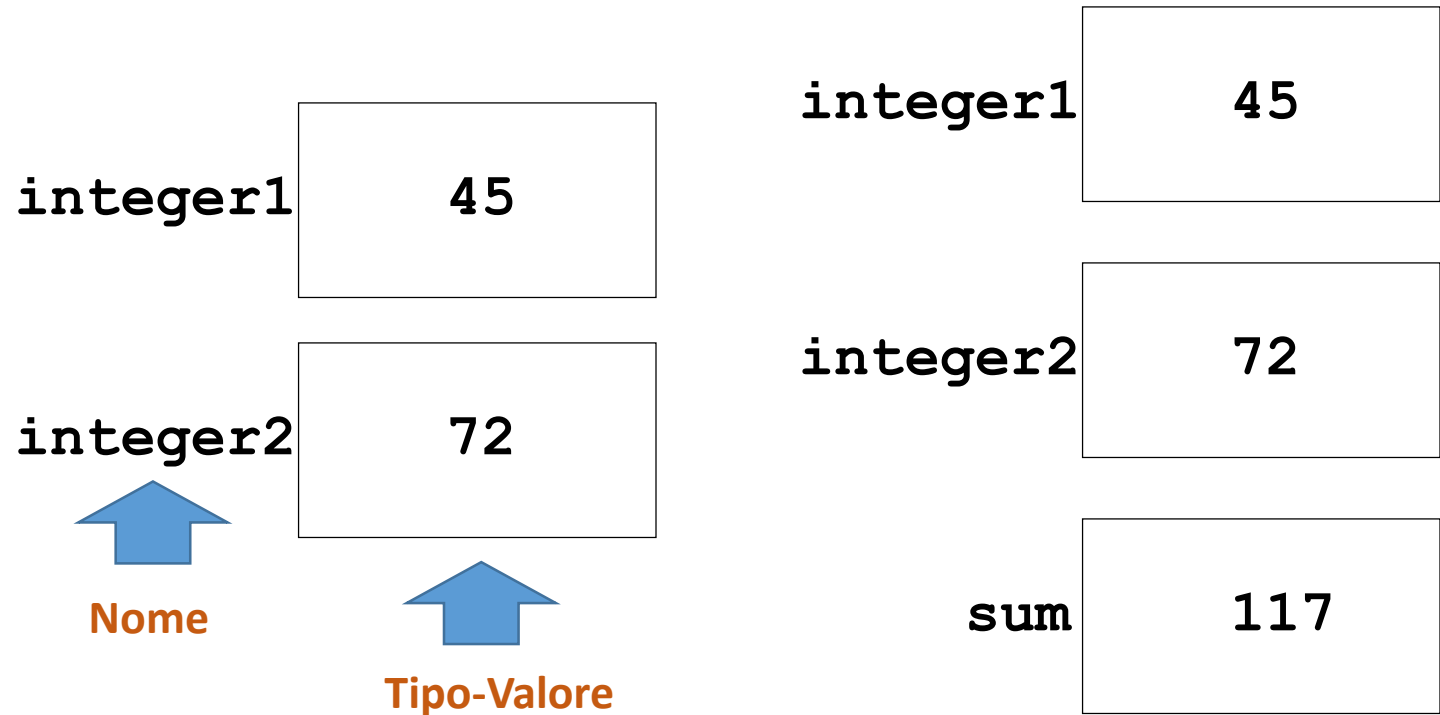
Ogni variabile è identificata attraverso tre elementi. **Quali?**

- **Nome**
- **Tipo**
- **Valore**

Quando dichiariamo una variabile è importante **inizializzarla**. **Perché?**

- Perché un nome di variabile è un alias di una locazione di memoria
- Inizializzando la variabile sovrascriviamo il valore memorizzato in quella locazione di memoria

Linguaggio C: cenni sulla memoria

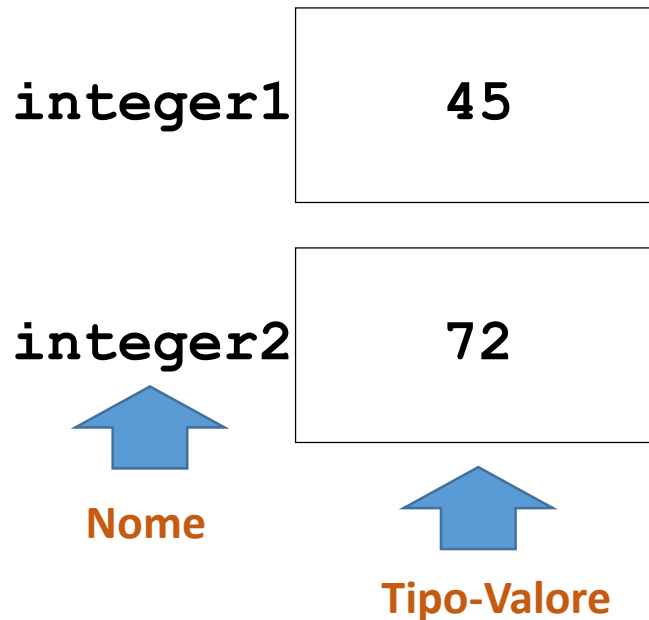


Ogni variabile è identificata attraverso tre elementi. **Quali?**

- **Nome**
- **Tipo**
- **Valore**

Quando assegniamo un valore bisogna **usare l'operatore &**, per indicare di memorizzare il valore **nella locazione di memoria cui punta la variabile**.

Linguaggio C: cenni sulla memoria



- Il tipo di dato determina **il range e la tipologia di valori** che una variabile può assumere
 - La scelta del tipo di dato più corretto per una variabile è **una importante scelta progettuale** in fase di risoluzione del problema e codifica dell'algoritmo
 - Ad esempio le variabili di tipo “**short int**” possono assumere valori tra -32768 e 32767 (65536 valori in tutto)
 - **Perchè esattamente 65536?**

Linguaggio C: tipi di dato

Perché le variabili intere di tipo short int occupano 16 bit, quindi possono assumere 2^{16} diverse combinazioni

La scelta del tipo di dato da associare a una variabile deve essere operata valutando **il dominio della variabile** (l'insieme dei valori che può assumere).

Variabili più piccole riducono la quantità di memoria richiesta dal programma, ma bisogna assicurarsi che i valori da memorizzare nella variabile siano adeguati al tipo di dato scelto.



Linguaggio C: tipi di dato

Tipo di dato	Dimensione in Byte (Macchina a 32 bit)	Dimensione in Byte (Macchina a 64 bit)	Range (Macchina a 64 bit)	
char	1	1	-128	127
unsigned char	1	1	0	255
short	2	2	-32768	32767
unsigned short	2	2	0	65535
int	4	4	-2147483648	2147483647
unsigned int	4	4	0	4294967295
long	4	8	-9223372036854775808	9223372036854775808
unsigned long	4	8	0	18446744073709551615
float	4	4
double	8	8
Long double	12	16

Linguaggio C: tipi di dato

Verificate da soli!

```
#include <stdio.h>

int main(void) {
    char charVariable;                unsigned char unsignedCharVariable;  short shortVariable;
    unsigned short unsignedShortVariable;  int integerVariable;          unsigned int unsignedIntVariable;
    long longVariable;                unsigned long unsignedLongVariable;
    float floatVariable;              double doubleVariable;        long double longDoubleVariable;

    printf("Size of char: %d \n", sizeof(charVariable));
    printf("Size of unsigned char: %d \n", sizeof(unsignedCharVariable));
    printf("Size of short: %d \n", sizeof(shortVariable));
    printf("Size of unsigned short: %d \n", sizeof(unsignedShortVariable));
    printf("Size of integer: %d \n", sizeof(integerVariable));
    printf("Size of unsigned integer: %d \n", sizeof(unsignedIntVariable));
    printf("Size of long: %d \n", sizeof(longVariable));
    printf("Size of unsigned long: %d \n", sizeof(unsignedLongVariable));
    printf("Size of float: %d \n", sizeof(floatVariable));
    printf("Size of double: %d \n", sizeof(doubleVariable));
    printf("Size of long double: %d \n", sizeof(longDoubleVariable));
}
```

Linguaggio C: tipi di dato

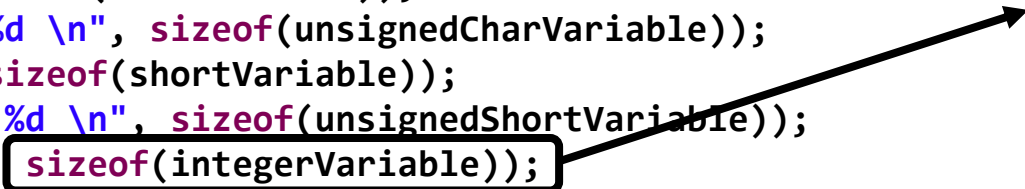
Verificate da soli!

```
#include <stdio.h>

int main(void) {
    char charVariable;
    unsigned short unsignedShortVariable;
    long longVariable;
    float floatVariable;

    unsigned char unsignedCharVariable;
    short shortVariable;
    int integerVariable;
    unsigned int unsignedIntVariable;
    unsigned long unsignedLongVariable;
    double doubleVariable;
    long double longDoubleVariable;

    printf("Size of char: %d \n", sizeof(charVariable));
    printf("Size of unsigned char: %d \n", sizeof(unsignedCharVariable));
    printf("Size of short: %d \n", sizeof(shortVariable));
    printf("Size of unsigned short: %d \n", sizeof(unsignedShortVariable));
    printf("Size of integer: %d \n", sizeof(integerVariable));
    printf("Size of unsigned integer: %d \n", sizeof(unsignedIntVariable));
    printf("Size of long: %d \n", sizeof(longVariable));
    printf("Size of unsigned long: %d \n", sizeof(unsignedLongVariable));
    printf("Size of float: %d \n", sizeof(floatVariable));
    printf("Size of double: %d \n", sizeof(doubleVariable));
    printf("Size of long double: %d \n", sizeof(longDoubleVariable));
}
```



L'operatore **sizeof()** restituisce la dimensione di una variabile

**Programming
is thinking, not typing**

Problem Solving

Programming
is thinking, not typing

- **Prima** di scrivere un programma
 - Comprendere a **fondo il problema** (**analisi**)
 - Pianificare con cura un **approccio per risolverlo** (approccio **top-down, bottom-up**)
 - Produrre una soluzione in pseudo-codice o con I flow-chart

Problem Solving

Programming
is thinking, not typing

- **Prima** di scrivere un programma
 - Comprendere a **fondo il problema** (**analisi**)
 - Pianificare con cura un **approccio per risolverlo** (approccio **top-down, bottom-up**)
 - Produrre una soluzione in pseudo-codice o con I flow-chart
- **Mentre** scrivete un programma
 - Individuate quali “building blocks” sono disponibili (**riuso** del codice)
 - La maggior parte dei programmi segue una struttura “standard”
 - **Definizione e inizializzazione** delle variabili (acquisizione dell’input)
 - **Elaborazione** dei dati
 - **Visualizzazione** dei risultati (output)

Programmazione Strutturata

- **Teorema di Bohm e Jacopini:** tutti i programmi possono essere scritti usando tre strutture di controllo fondamentali
 - **Sequenza:**
 - Nativa nel C. I programmi vengono eseguiti sequenzialmente per default
 - **Selezione:**
 - Il C ne ha tre tipi: if, if...else, switch
 - **Iterazione:**
 - Il C ne ha tre tipi: while, do...while e for



Struttura di Selezione

Usata per scegliere tra diverse alternative

Istruzione	Pseudocodice	Traduzione in C
if		
if...else		

**Scrivere un programma
che stampi un messaggio
differente a seconda che
il voto ricevuto sia
maggiore o minore di 18**

Struttura di Selezione

Usata per scegliere tra diverse alternative

Istruzione	Pseudocodice	Traduzione in C
if	Se il voto dello studente è maggiore di 18 Stampa "Promosso"	
if...else	Se il voto dello studente è maggiore di 18 Stampa "Promosso" Altrimenti Stampa "Bocciato"	

Struttura di Selezione


Usata per scegliere tra diverse alternative

Istruzione	Pseudocodice	Traduzione in C
if	Se il voto dello studente è maggiore di 18 Stampa "Promosso"	<pre>if (voto >= 18) puts("Promosso\n");</pre>
if...else	Se il voto dello studente è maggiore di 18 Stampa "Promosso" Altrimenti Stampa "Bocciato"	<pre>if (voto >= 18) { puts("Promosso\n"); } else puts("Bocciato\n");</pre> <p>Oppure</p> <pre>voto >= 18 ? puts("Promosso\n") : puts("Bocciato\n");</pre>

Importante: l'indentazione rende il programma **più leggibile**


Struttura di Selezione

Usata per scegliere tra diverse alternative

Istruzione	Pseudocodice	Traduzione in C
switch	<p>Se il voto dello studente è maggiore di 18 Stampa "Promosso" Altrimenti Stampa "Bocciato"</p>  <p>Lo pseudocodice è analogo a quella dell'istruzione if...else ma si esprime in modo diverso.</p>	


Struttura di Selezione

Usata per scegliere tra diverse alternative

Istruzione	Pseudocodice	Traduzione in C
switch	<p>Se il voto dello studente è maggiore di 18 Stampa "Promosso" Altrimenti Stampa "Bocciato"</p>  <p>Lo pseudocodice è analogo a quella dell'istruzione if...else ma si esprime in modo diverso.</p>	<pre>switch(voto) { case 0: case 1: case 2: case 3: case 4: case 5: case 6: case 7: case 8: case 9: case 10: case 11: case 12: case 13: case 14: case 15: case 16: case 17: puts("Bocciato\n"); break; default: puts("Promosso\n"); }</pre>

Struttura di Selezione

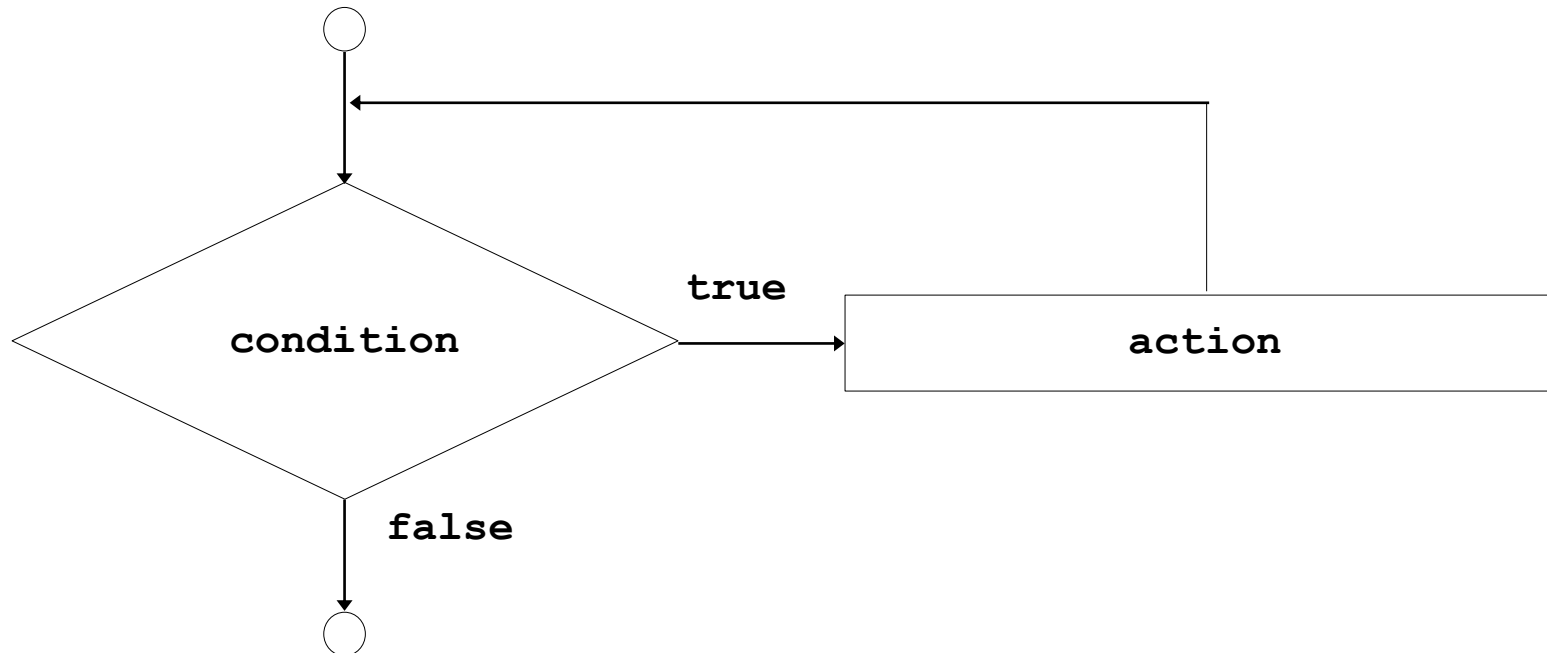
Usata per scegliere tra diverse alternative

Istruzione	Pseudocodice	Traduzione in C
switch	<p>Se il voto dello studente è maggiore di 18 Stampa "Promosso" Altrimenti Stampa "Bocciato"</p>  <p>Lo pseudocodice è analogo a quella dell'istruzione if...else ma si esprime in modo diverso.</p>	<pre>switch(voto) { case 0: case 1: case 2: case 3: case 4: case 5: case 6: case 7: case 8: case 9: case 10: case 11: case 12: case 13: case 14: case 15: case 16: case 17: puts("Bocciato\n"); break; default: puts("Promosso\n"); }</pre>

E' una tipologia di **istruzione molto utile** se dobbiamo valutare **diverse alternative di tipo categorico** (non numerico!) – **es:** città di nascita, provincia di residenza, voto di un esame (in lettere, tipo 'A', 'B', 'C'), gruppo sanguigno, etc.

Struttura di Iterazione - `while`

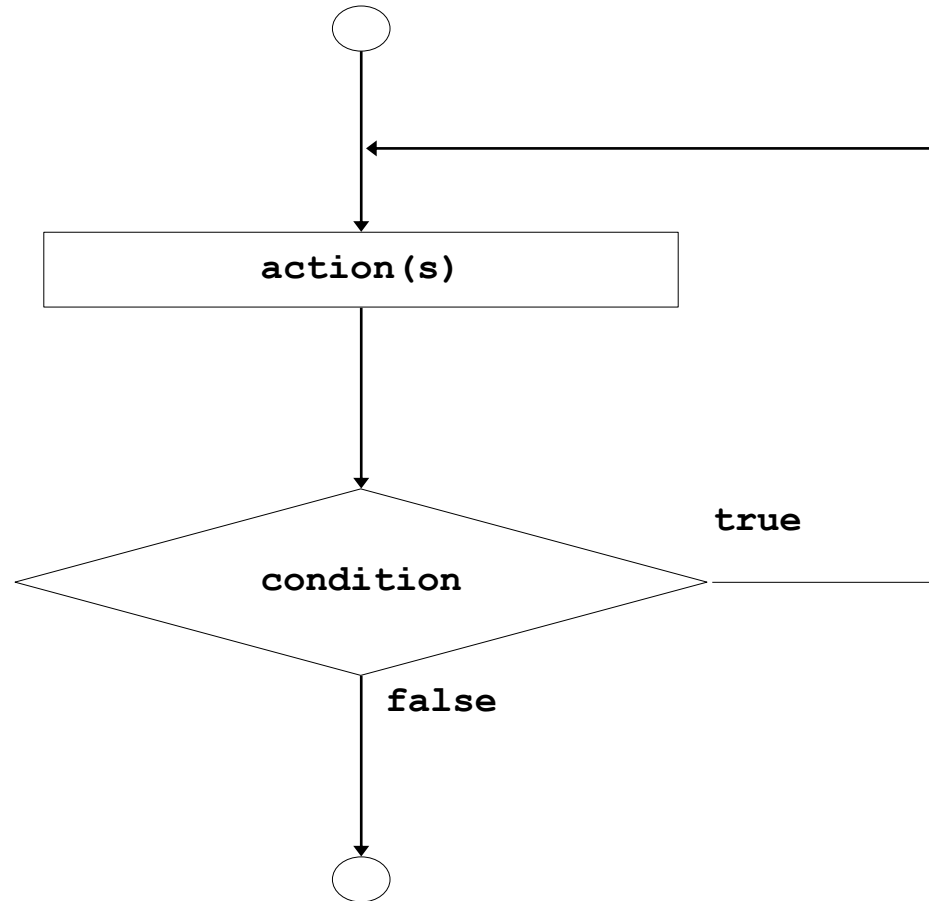
Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera



Ripete le **istruzioni contenute nel ciclo** finchè la condizione è vera.

Struttura di Iterazione – `do . . while`

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

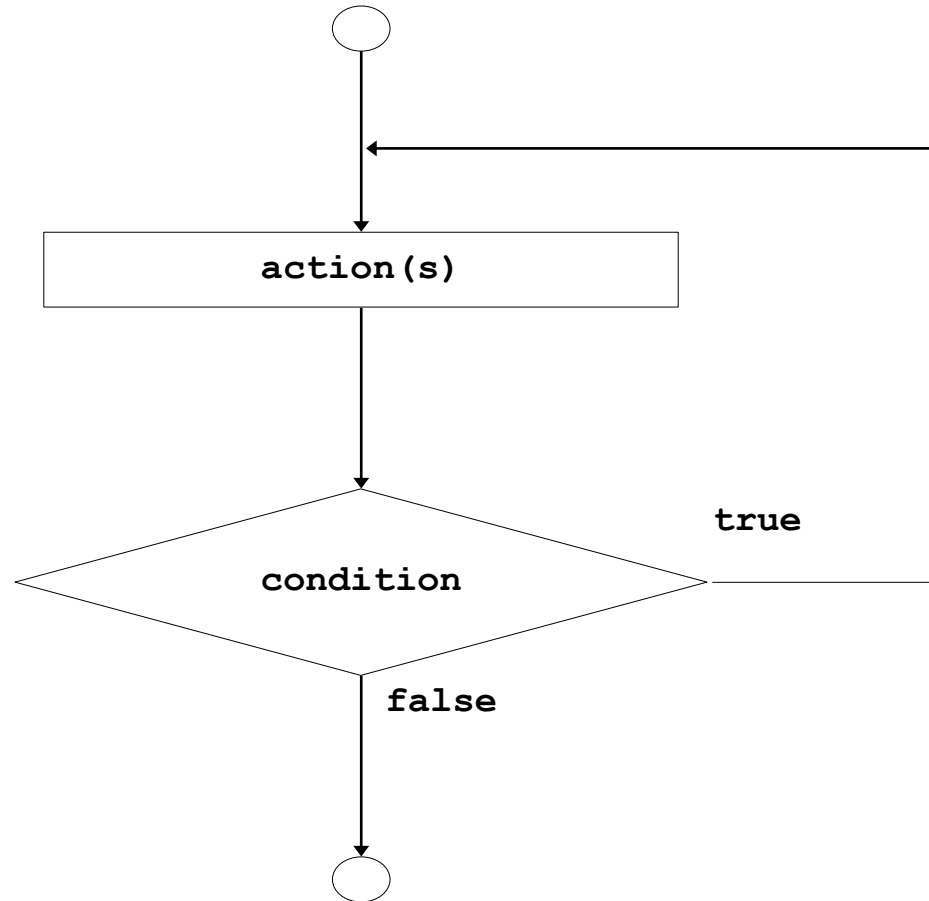


Ripete le **istruzioni contenute nel ciclo** finchè la condizione è vera.

Qual è la differenza?

Struttura di Iterazione – `do..while`

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

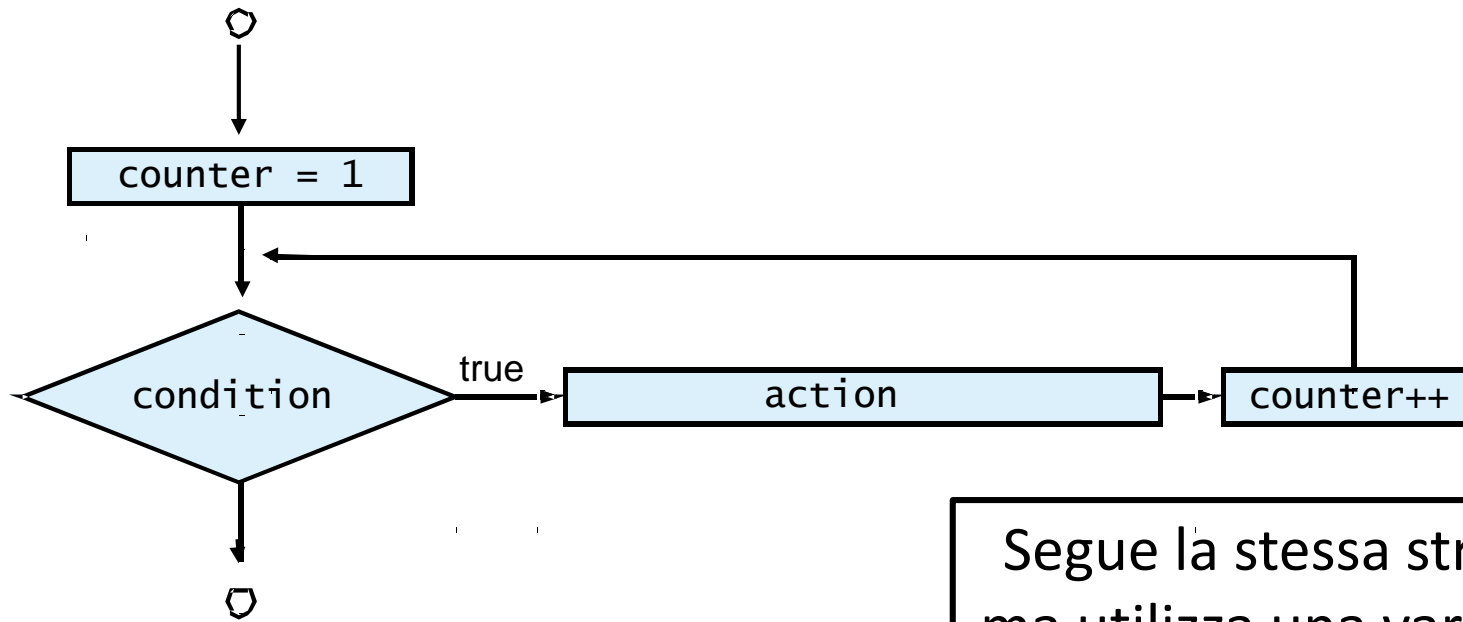


Ripete le **istruzioni contenute nel ciclo** finchè la condizione è vera.

IMPORTANTE: Esegue almeno **un ciclo!**

Struttura di Iterazione – for

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.



Segue la stessa struttura del **while**,
ma utilizza una variabile contatore che
**viene incrementata dopo il blocco
delle istruzioni (action)**

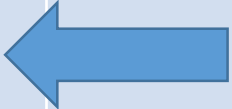
Struttura di Iterazione

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while		Scrivere un programma che conteggi il costo totale dei prodotti in un carrello. Sappiamo che il carrello può contenere esattamente cinque prodotti.
do..while		
for		

Struttura di Iterazione

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	 <div>Scrivere un programma che conteggi il costo totale dei prodotti in un carrello. Sappiamo che il carrello può contenere esattamente cinque prodotti.</div>
do..while		
for		

Struttura di Iterazione

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	<div>Scrivere un programma che conteggi il costo totale dei prodotti in un carrello. Sappiamo che il carrello può contenere esattamente cinque prodotti.</div>
do..while	ripeti leggi costo aggiungi il costo al totale finchè (numero_prodotti<5)	
for	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	



Struttura di Iterazione

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	<div>Scrivere un programma che conteggi il costo totale dei prodotti in un carrello. Sappiamo che il carrello può contenere esattamente cinque prodotti.</div>
do..while	ripeti leggi costo aggiungi il costo al totale finchè (numero_prodotti<5)	
for	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	



Struttura di Iterazione

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	
do..while	ripeti leggi costo aggiungi il costo al totale finchè (numero_prodotti<5)	<div>Come lo rendiamo in linguaggio C?</div>
for	finchè (numero_prodotti<5) leggi costo aggiungi il costo al totale	

Struttura di Iterazione

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (numero_prodotti<=5) leggi costo aggiungi il costo al totale	<pre>while(products<5){ scanf(«%d»,&costo); totale = totale + costo; products++; }</pre>
do..while	ripeti leggi costo aggiungi il costo al totale finchè (numero_prodotti<=5)	<pre>do { scanf(«%d»,&costo); totale = totale + costo; products++; } while(products<5);</pre>
for	finchè (numero_prodotti<=5) leggi costo aggiungi il costo al totale	<pre>for(products=0; products<5; products++) { scanf(«%d»,&costo); totale = totale + costo; }</pre>

Problema 1.1

Scrivere un programma che conteggi il costo totale dei prodotti in un carrello.
Non sappiamo quanti prodotti può contenere il carrello.

Input?

Costo dei singoli prodotti

Output?

Costo totale della spesa

Quale tipologia di istruzioni ci serve? Perché?

Struttura di **iterazione**. Perché il programma effettua una operazione (ciclica) sommando il costo dei prodotti nel carrello. **Non conosciamo a priori il numero di prodotti da inserire.**

Problema 1.1

Scrivere un programma che conteggi il costo totale dei prodotti in un carrello.
Non sappiamo quanti prodotti può contenere il carrello.

Input?

Costo dei singoli prodotti

Output?

Costo totale della spesa

Quale tipologia di istruzioni ci serve? Perché?

Struttura di **iterazione**. Perché il programma effettua una operazione (ciclica) sommando il costo dei prodotti nel carrello. Non conosciamo a priori il numero di prodotti da inserire.

E' necessario introdurre il concetto di **iterazione non controllata**

Problema 1.1

Scrivere un programma che conteggi il costo totale dei prodotti in un carrello.
Non sappiamo quanti prodotti può contenere il carrello.

Input?

Costo dei singoli prodotti

Output?

Costo totale della spesa

Quale tipologia di istruzioni ci serve? Perché?

Struttura di **iterazione**. Perché il programma effettua una operazione (ciclica) sommando il costo dei prodotti nel carrello. Non conosciamo a priori il numero di prodotti da inserire.

E' necessario introdurre il concetto di **iterazione non controllata!**

Non si utilizza un contatore fisso, si utilizza un valore sentinella

Se il valore letto è uguale al valore sentinella, esce dal ciclo.

Struttura di Iterazione (non controllata)

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>while(costo != -1) { scanf(«%d», &costo); totale = totale + costo; }</pre>
Do..while	ripeti leggi costo aggiungi il costo al totale finchè (sentinella=true)	<pre>do { scanf(«%d», &costo); totale = totale + costo; } while(costo != -1);</pre>
for	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>for(costo = 0; costo != -1;) { scanf(«%d», &costo); totale = totale + costo; }</pre>

Struttura di Iterazione (non controllata)

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>while(costo != -1){ scanf(«%d», &costo); totale = totale + costo; }</pre>
Do..while	ripeti leggi costo aggiungi il costo al totale finchè (sentinella=true)	<pre>do { scanf(«%d», &costo); totale = totale + costo; } while(costo != -1);</pre>
for	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>for(costo = 0; costo != -1;) { scanf(«%d», &costo); totale = totale + costo; }</pre>

Il valore sentinella va nella condizione. Si tendono ad utilizzare **valori non validi** per quella variabile

Struttura di Iterazione (non controllata)

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>while(costo != -1){ scanf(«%d», &costo); totale = totale + costo; }</pre>
Do..while	ripeti leggi costo aggiungi il costo al totale finchè (sentinella=true)	<pre>do { scanf(«%d», &costo); totale = totale + costo; } while(costo != -1);</pre>
for	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>for(costo = 0; costo != -1;) { scanf(«%d», &costo); totale = totale + costo; }</pre>

**Soluzione poco
leggibile e poco
interpretabile**

Struttura di Iterazione (non controllata)

Utilizzata per esprimere operazioni che si ripetono finchè una determinata condizione resta vera.

Istruzione	Pseudocodice	Traduzione in C
while	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>while(costo != -1){ scanf(«%d», &costo); totale = totale + costo; }</pre>
Do..while	ripeti leggi costo aggiungi il costo al totale finchè (sentinella=true)	<pre>do { scanf(«%d», &costo); totale = totale + costo; } while(costo != -1);</pre>
for	finchè (sentinella=true) leggi costo aggiungi il costo al totale	<pre>for(costo = 0; costo != -1;) { scanf(«%d», &costo); totale = totale + costo; }</pre>

Suggerimento: **while** e **do...while** si tendono a preferire per le iterazioni non controllate, mentre il **for** è la struttura più semplice da adottare quando sappiamo a priori il numero di iterazioni da eseguire è noto.

Problema 1.1

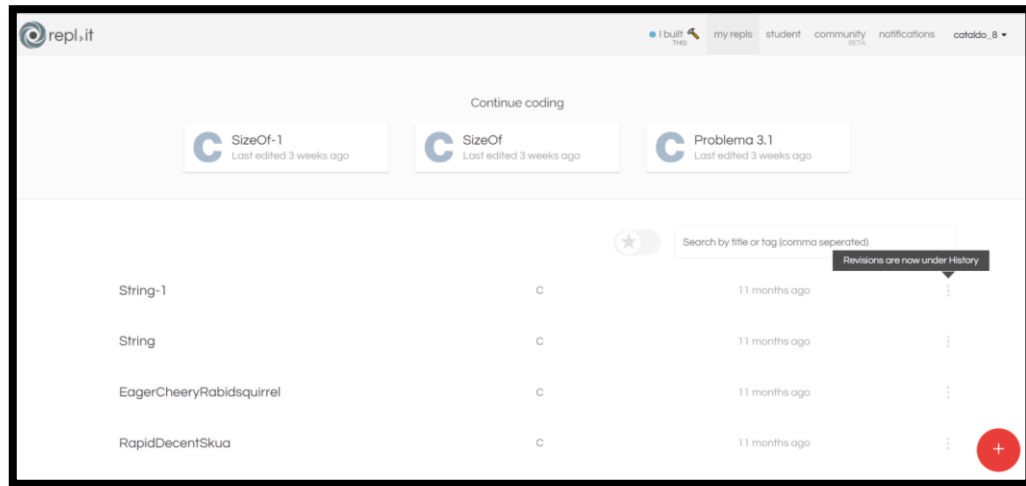
Scrivere un programma che conteggi il costo totale dei prodotti in un carrello.
Non sappiamo quanti prodotti può contenere il carrello.

Proviamo a codificare una soluzione su Repl.it

Problema 1.1

Scrivere un programma che conteggi il costo totale dei prodotti in un carrello.
Non sappiamo quanti prodotti può contenere il carrello.

Proviamo a codificare una soluzione su Repl.it



**(My Repls → Simbolo '+' in basso →
Linguaggio C)**

Soluzione (con `while` + valore sentinella)

```
1    #include "stdio.h"
2
3    int main(void) {
4        int costo = 0;           // variabile di tipo int
5        int totale = 0;
6
7        while(costo != -1) {
8            printf("Inserisci il costo del prodotto: ");
9            scanf("%d", &costo);
10           totale = totale + costo;
11        }
12
13        printf("Totale: %d", totale);
14    }
```

Soluzione (con `while` + valore sentinella)

```
1  #include "stdio.h"
2
3  int main(void) {
4      int costo = 0;           // variabile di tipo int
5      int totale = 0;
6
7      while(costo != -1) {
8          printf("Inserisci il costo del prodotto: ");
9          scanf("%d", &costo);
10         totale = totale + costo;
11     }
12
13     printf("Totale: %d", totale);
14 }
```

**E' una soluzione
corretta?**

Soluzione (con `while` + `do-while`)

```
1  #include "stdio.h"
2
3  int main(void) {
4      int costo = 0;
5      int totale = 0;
6
7      while(costo != -1) {
8          printf("Inserisci il costo del prodotto: ");
9          scanf("%d", &costo);
10         totale = totale + costo;
11     }
12
13     printf("Totale: %d", totale);
14 }
```

```
gcc version 4.6.3
Inserisci il costo del prodotto: 2
Inserisci il costo del prodotto: 3
Inserisci il costo del prodotto: 6
Inserisci il costo del prodotto: -1
Totale: 10
```

**No, perché il valore
sentinella viene
comunque aggiunto
al totale**

Soluzione (con `while` + `\n`)

```
1    #include "stdio.h"
```

```
gcc version 4.6.3
```

```
Inserisci il costo del prodotto: 2
Inserisci il costo del prodotto: 3
Inserisci il costo del prodotto: 6
Inserisci il costo del prodotto: -1
```

Bisogna applicare i principi della programmazione difensiva: il programma deve funzionare correttamente anche davanti a input non corretti

```
10    totale = totale + costo,
11    }
12
13    printf("Totale: %d", totale);
14 }
```

sentinella viene comunque aggiunto al totale

Soluzione (con `while` + valore sentinella)

```
1  #include "stdio.h"
2
3  int main(void) {
4      int costo = 0;                // variabile di tipo int
5      int totale = 0;
6
7      printf("Inserisci il costo del prodotto: ");
8      scanf("%d", &costo);
9
10     while(costo != -1) {
11         totale = totale + costo;
12         printf("Inserisci il costo del prodotto: ");
13         scanf("%d", &costo);
14     }
13     printf("Totale: %d", totale);
14 }
```

**Programma
(parzialmente)
Corretto**

Soluzione (con `while` + valore sentinella)

```
1    #include "stdio.h"
2
3    int main(void) {
4        int costo = 0;                // variabile di tipo int
5        int totale = 0;
6
7        printf("Inserisci il costo del prodotto: ");
8        scanf("%d", &costo);
9
10       while(costo != -1) {
11           totale = totale + costo;
12           printf("Inserisci il costo del prodotto: ");
13           scanf("%d", &costo);
14       }
13       printf("Totale: %d", totale);
14   }
```

**Se il primo input inserito
è -1 il programma non
entra nel ciclo e stampa
zero: CORRETTO**

Soluzione (con `while` + valore sentinella)

```
1    #include "stdio.h"
2
3    int main(void) {
4        int costo = 0;                // variabile di tipo int
5        int totale = 0;
6
7        printf("Inserisci il costo del prodotto: ");
8        scanf("%d", &costo);
9
10       while(costo != -1) {
11           totale = totale + costo;
12           printf("Inserisci il costo del prodotto: ");
13           scanf("%d", &costo);
14       }
13       printf("Totale: %d", totale);
14   }
```

**Mancano ancora dei
controlli sul valore
inserito per la variabile
«costo»**

Domande?

Laboratorio di Informatica
docente: Cataldo Musto
cataldo.musto@uniba.it



Esercitazione 0

- **Obiettivo:** utilizzare i concetti base di programmazione strutturata, combinando **istruzioni di iterazione e selezione** per risolvere problemi complessi. Dimostrare anche di avere conoscenza relativa alla **tipizzazione**.
- *Assignment su Repl.it*
 - Esercitazione di autovalutazione. Non sarà assegnato alcun voto.

Esercitazione 0

Il Body Mass Index (BMI) è calcolato come il rapporto tra il peso e il quadrato dell'altezza (espresso in metri) di un individuo. Gli individui sovrappeso hanno un BMI superiore a 25

$$\text{BMI} = \text{peso} / (\text{altezza_in_metri} * \text{altezza_in_metri})$$

1. **Scrivere un algoritmo che generi random età, altezza e peso di dieci individui e calcoli il BMI di ciascuno di essi e lo mostri sullo schermo.**
2. **L'algoritmo deve calcolare il massimo di altezza e peso e la media del BMI , distinguendo over 40 e under 40**
3. **L'algoritmo deve anche stampare in output i valori massimi di altezza e peso (per le due categorie)**
4. **L'algoritmo deve stampare un messaggio sullo schermo del tipo «Il campione è mediamente sovrappeso» oppure «il campione è mediamente sottopeso» a seconda che il BMI medio sia maggiore o minore di 25 (sia per gli under 40 che per gli over).**

Esercitazione 0 (cont.)

- **INPUT**

- Età, Altezza, Peso, generati random per 10 individui
 - **Attenzione a generare valori in un intervallo corretto!**

- **OUTPUT**

- Altezza massima over 40, Altezza massima under 40
- Peso massimo over 40, Peso massimo under 40
- BMI medio over 40, BMI medio under 40

- **NOTE**

- **Porre attenzione allo stile di programmazione:** nomenclatura, costanti simboliche e indentazione
- Porre attenzione a **scrivere commenti significativi**
- Porre attenzione alla **programmazione difensiva**: il programma funziona anche nei casi "limite" ?

Esercitazione 0 (cont.)

• SUGGERIMENTI

Per la generazione di un numero random in un range tra MIN e MAX utilizzare il seguente frammento di codice

```
#include "stdlib.h" // libreria che include la funzione rand()
#include "time.h" // libreria per randomizzare il seed
int seed = time(NULL);
srand(seed);
int randomValue = rand() % (MAX-MIN+1) + MIN;
```

Se vogliamo generare un numero tra 10 e 30 la funzione sarà

```
int randomValue = rand() % 21 + 10;
```

Perchè l'operatore % ci restituisce un valore tra 0 e 20, cui sommeremo 10 che è il valore minimo possibile.

Esercitazione 0 (cont.)

- **ESTENSIONI**

Modificare l'algoritmo utilizzando un valore sentinella invece di un ciclo con valore predefinito

Svolgete l'assignment su Repl.it

Termine ultimo: Sabato 3 Marzo