Primi passi…

- *"Il mio primo programma…"*

```
#include <stdio.h>
/* l'esecuzione comincia dalla funzione main */
int main()
{
  printf( "Hello World!!\n" );
  return 0; /* il programma termina con successo */
} /* fine del blocco di codice costituente il main */
```

1

Primi passi…

- **`#include <stdio.h>`**
  - Direttiva di preprocessore: include le dichiarazioni delle funzioni per l'input/output – il linker assocerà dichiarazioni a definizioni
- **`int main()`**
  - I programmi C contengono una o più funzioni (cosa sono ?!), una delle quali **deve essere** main
  - Le parentesi sono usate per indicare una funzione
  - **`int`** significa che main "ritorna" un valore intero
  - Le parentesi graffe ({ e }) indicano un blocco
  - Il corpo di tutte le funzioni deve essere contenuto tra parentesi graffe

2

Primi passi…

- `printf( "Hello world!!\n" );`
  - Istruzione → esegue una certa azione
    - Stampa la stringa di caratteri all'interno delle virgolette (" ")
  - L'intera linea è chiamata statement
    - Tutti gli statements devono terminare con un punto e virgola (;)
  - Caratteri di escape (\)
    - indicano che la printf deve fare qualcosa fuori dall'ordinario
    - \n è un carattere di nuova linea

3

Primi passi…

- return 0;
  - Un modo per uscire da una funzione
  - return 0, in questo caso significa che il programma è terminato normalmente
- Parentesi graffa destra }
  - Indica che la fine del main è stata raggiunta
- Linker
  - Quando una function viene chiamata, il linker la localizza nella libreria
  - La inserisce nel programma oggetto
  - Se il nome della funzione è scritto in modo errato, il linker produrrà un errore perchè non sarà in grado di trovare la funzione nella libreria

4

Secondi passi…

- Somma di due numeri interi
  - …è abbastanza facile?
- Definizione dell'algoritmo

1. Lettura dei valori da sommare
2. Esecuzione della somma
3. Stampa del risultato

5

Somma

- Predisposizione delle variabili necessarie

```
int intero1;
int intero2;
int somma;
```

- Lettura dei valori da sommare

```
scanf("%d", &intero1);
scanf("%d", &intero2);
```

- Notare il passaggio per riferimento!!!
  - Sì, ok, ma che roba è?!

6

Somma

- Esecuzione della somma

```
somma = intero1 + intero2;
```

- Stampa del risultato

```
printf("%d + %d = %d", intero1, intero2,
  somma);
```

- Terminazione del programma

```
return 0;
```

7

Somma: all together now!

```
int main()
{
  int intero1, intero2, somma;
  printf("Inserire due valori:\n");
  scanf("%d", &intero1);
  scanf("%d", &intero2);
  somma = intero1 + intero2;
  printf("\n%d + %d = %d", intero1, intero2,
     somma);
  return 0;
}
```

8

Somma: commenti

- **`int intero1, intero2, somma;`**
  - Definizione di variabili
  - Variabili: locazioni in memoria dove è possibile memorizzare un valore
  - **`int`** significa che le variabili possono contenere interi (-1, 3, 0, 47)
  - Nomi di variabili (identificatori)
    - **`intero1, intero2, somma`**
    - Identificatori: consistono di lettere, cifre (non possono cominciare con una cifra) e underscore (_)
  - Case sensitive
  - Le definizioni appaiono prima degli statement che le utilizzano
    - Se uno statement referenzia una variabile non dichiarata, sarà prodotto un errore sintattico da parte del compilatore

9

Somma: commenti

- **`scanf( "%d", &intero1 );`**
  - Ottiene un valore dall'utente
    - **`scanf`** usa lo standard input (generalmente la tastiera)
  - **`scanf`** ha due argomenti (parametri)
    - **`%d`** – indica che il dato dovrebbe essere un intero decimale
    - **`&intero1`** - locazione in memoria per memorizzare la variable
    - **`&`** inizialmente provoca confusione – per ora, ricordate solo di includerlo per i nomi delle variabili nello statement scanf
  - Durante l'esecuzione del programma l'utente risponde alla scanf inserendo un numero, e poi premendo il tasto *enter* (return)

10

Somma: commenti

- **=** (operatore di assegnamento)
  - Assegna un valore ad una variabile
  - E' un operatore binario (ha due operandi: l-value, r-value)
    - `somma = intero1 + intero2;`
    - **`somma`** avrà valore **`intero1 + intero2`**;
  - La variable a sinistra riceve il valore

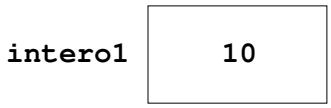
- `printf( "Sum is %d\n", somma );`
  - Simile alla **`scanf`**
    - **`%d`** indica che un intero decimale sarà stampato
    - **`somma`** specifica quale intero sarà stampato
  - I calcoli posso essere eseguiti all'interno della **`printf`**

```
printf( "Sum is %d\n", intero1 + intero2 );
```

11

Le variabili

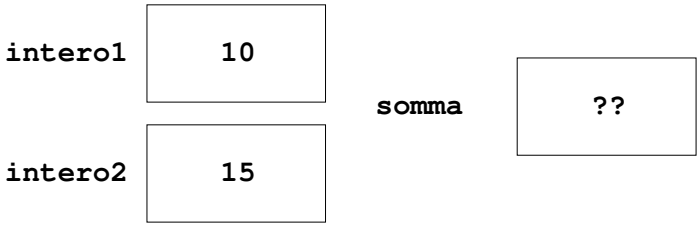
- I nomi delle variabili corrispondono a locazioni nella memoria del computer
- Ogni variabile è caratterizzata da un nome, un tipo, una dimensione (??) ed un valore
- L'inserimento di un nuovo valore in una variabile (ad esempio attraverso una **`scanf`**), rimpiazza e distrugge il valore precedente
- La lettura (l'utilizzo) di variabili dalla memoria non cambia il loro valore



12

Le variabili

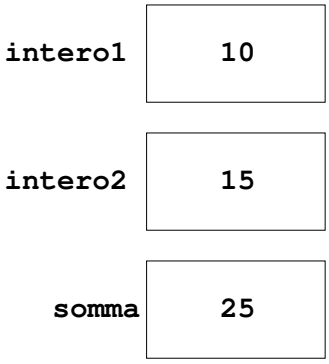
- Prima della dichiarazione non esistono
- Dopo la dichiarazione sono non inizializzate → il valore che contengono è aleatorio…
- Dopo la lettura (scanf) le variabili **`interox`** sono inizializzate al valore letto, **`somma`** è ancora non inizializzata



13

Le variabili

- Dopo l'esecuzione della somma e dopo la stampa (anche la stampa a video è una semplice lettura)



14

Note sull'aritmetica

- Attenzione a
  - Precedenza degli operatori
  - Associatività degli operatori

```
y = 2 * 8 / 4 + 4 * 5 + 1
z = 3 - 1 - 1
```

Come vengono valutate?

15

Precedenza e associatività

- Ogni operatore nel set di operatori supportato dall'analizzatore di espressioni ha una precedenza e prevede una direzione di valutazione
- La direzione di valutazione di un operatore è l'associatività dell'operatore.
- Gli operatori con precedenza superiore vengono valutati prima di quelli con precedenza inferiore → Se un'espressione complessa include più operatori, l'ordine di esecuzione è determinato dalla precedenza degli operatori.
- Se un'espressione contiene più operatori con la stessa precedenza, gli operatori verranno valutati nell'ordine in cui compaiono, procedendo da sinistra a destra o da destra a sinistra a seconda della loro associatività

16

Operatori

Operatori	Simboli	Associatività
Chiamata a procedura Selezioni	() [] -> .	da sinistra a destra
Unari	! ~ + - ++ -- & * (type) sizeof	da destra a sinistra
Moltiplicativi	* / %	da sinistra a destra
Additivi	+ -	da sinistra a destra
Shift	<< >>	da sinistra a destra
Relazionali	< <= > >=	da sinistra a destra
Uguaglianza/Dis.	== !=	da sinistra a destra
AND bit a bit	&	da sinistra a destra
OR esclusivo bit a bit	^	da sinistra a destra
OR inclusivo bit a bit	|	da sinistra a destra
AND logico	&&	da sinistra a destra
OR logico	||	da sinistra a destra
Condizione	?:	da destra a sinistra
Assegnamenti	= += -= *= /= %= &= ^= |= <<= >>=	da destra a sinistra
Concatenazione	,	da sinistra a destra

17

Conversioni di tipo

- **Conversioni implicite** (←) e **promozioni di tipo** (↑) nelle espressioni:

```
double    ←     float
  ↑
unsigned long
  ↑
long
  ↑
unsigned int    ←   unsigned short
  ↑
int   ←    char, unsigned char, short, enum
```

18

Conversioni di tipo

- **Conversioni esplicite** (CASTING):
  `(nomeTipo)  espressione`
- Esempi

```
int v1, v2, v3;
float x, y;
v1 = 5;
v2 = 2;
x = v1 / v2;                    //  x = 2.0
y = (float) v1 / (float) v2;    //  y = 2.5
v3 = log(33);                   //  ??
```

19

Conversioni di tipo

```
int v3 = log(33);
```

- Si tenta di convertire un **`double`** in un **`int`**
- Viene segnalato come warning… ma è un errore!
  - Il compilatore C è molto sportivo (a volte anche troppo)
- Per avere una corretta conversione occorre un *cast* esplicito

```
int v3 = (int)log(40);
```

- Senza il *cast*, il **`double`** viene brutalmente interpretato come un **`int`**; il risultato è (quasi) impredicibile e dipende dal formato interno
- Con il **`cast`**, a **`v3`** viene assegnata la parte intera del logaritmo (nessun arrotondamento)

20

Note a margine

- **`log`** è una funzione che calcola il logaritmo in base **`e`**
- **`log10`** calcola il logaritmo in base 10
- …queste ed altre sono dichiarate nell'header file **`math.h`** e fanno parte della libreria standard di C

21