# L'assegnamento

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2012/2013

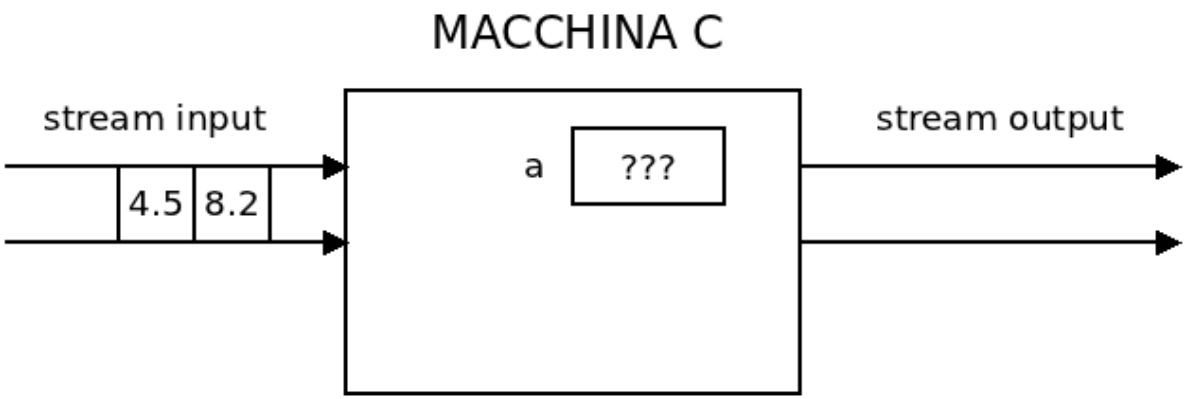
# Abbiamo visto

- È conveniente definire una macchina astratta C
- Lo stato della macchina ci dà tutte le informazioni sul suo funzionamento
  - Il valore assunto dalle variabili in un certo istante
  - Ciò che è presente sullo stream di input
  - Ciò che è stato scritto sullo stream di output
- L'esecuzione di un'istruzione altera lo stato della macchina astratta
  - La funzione di libreria `scanf` consuma un dato dall'input stream e lo trasferisce in una variabile in memoria codificandolo opportunamente
  - La funzione di libreria `printf` scrive sull'output stream. Può scrivere la rappresentazione simbolica del valore di una variabile.

# Cosa accade in questo caso?

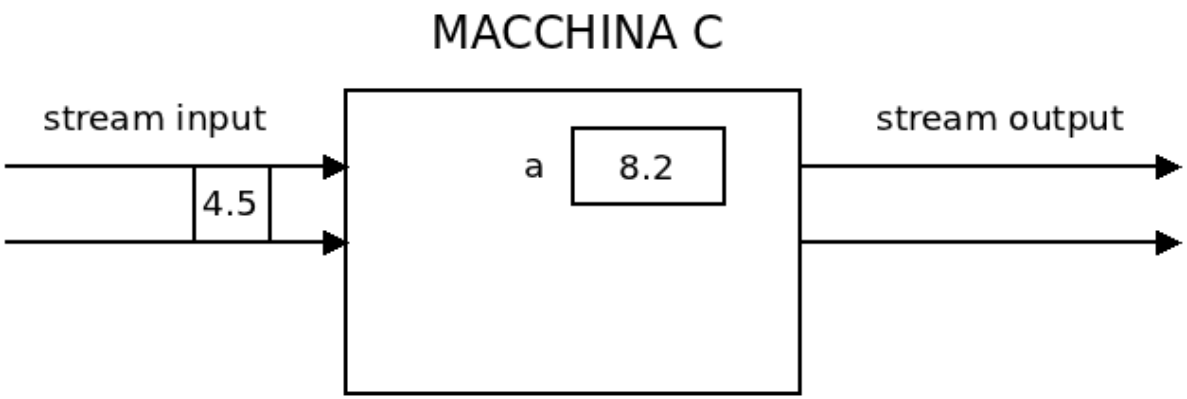
```
#include <stdio.h>
float a;

int main(){
    scanf(``%f'', &a);
    scanf(``%f'', &a);
    printf(``E stato inserito
            il valore %f \n'', a);
    return 0;
}
```

# Interprete



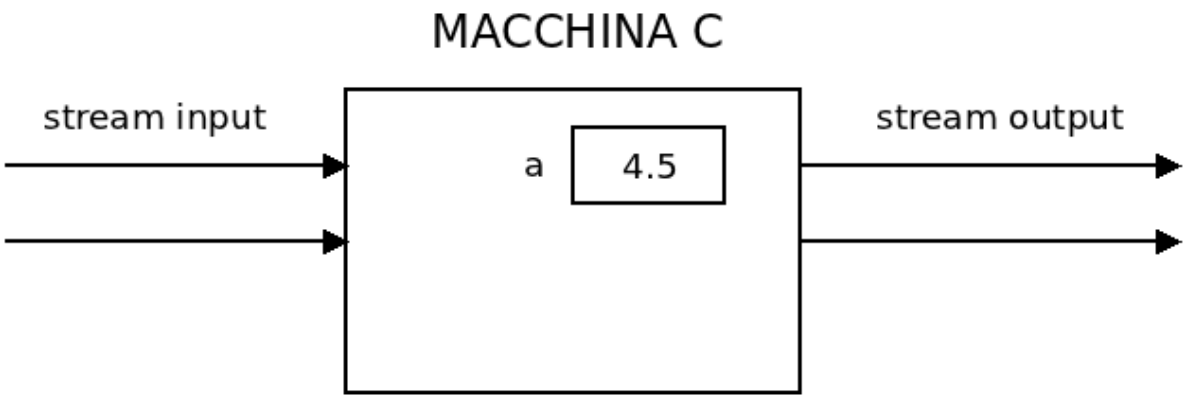

prossima istruzione:
scanf("%f",&a)

# Interprete



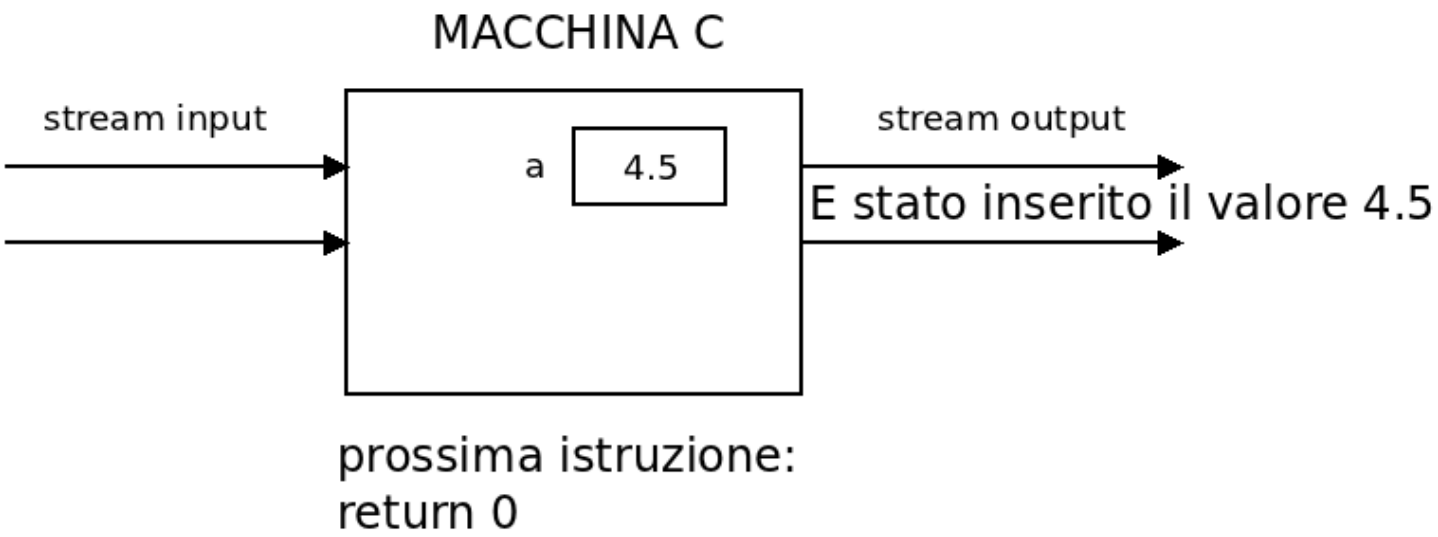

prossima istruzione:
scanf("%f",&a)

# Interprete




prossima istruzione:
printf("E stato inserito il valore %f",a)

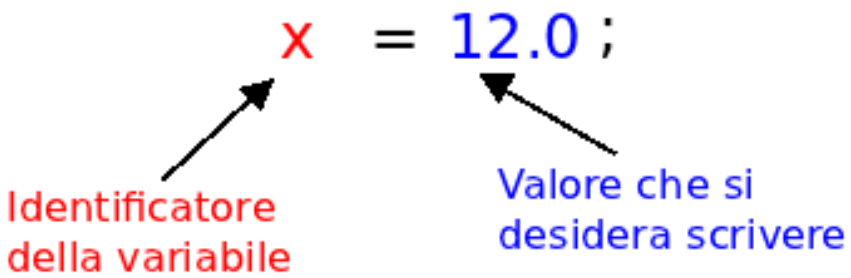
# Interprete




- Quando si scrive il valore di una variabile, il precedente valore è sovrascritto e non può essere recuperato dalla lettura della variabile in alcun modo

# Introduzione

- L'assegnamento è l'istruzione fondamentale dei linguaggi di programmazione imperativi
- Serve a scrivere un valore in una cella di memoria
- Il tipo del valore deve essere compatibile con il tipo della variabile
  - Nel seguente esempio `x` è di tipo `float` oppure `double`




- La `x` che sta a sinistra dell'assegnamento denota la locazione di memoria individuata dall'identificatore `x`
  - Si parla in questo caso di left-value di `x`

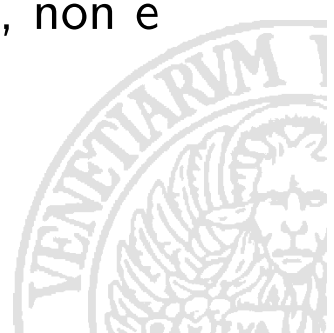
# Come specificare il valore da assegnare?

- Il valore da assegnare può essere il risultato di un *calcolo*
  - Naturalmente il risultato di questo calcolo deve avere tipo compatibile a quello della variabile
- Le espressioni servono a questo scopo
  - Un'espressione è una combinazione di operatori ed operandi. È caratterizzata da:
    - Un tipo, che è deciso staticamente (in fase di compilazione)
    - Un valore, che è deciso dinamicamente (in fase di esecuzione)

# Costanti e variabili

Nella sua forma piú semplice un'espressione può consistere di:

- Una costante il cui tipo e valore sono immediatamente noti
  - `1.73` è una costante di tipo `float` e il cui valore è 1.73
- Una variabile in questo caso il tipo dell'espressione è il tipo della variabile, e il valore dell'espressione è il valore memorizzato dalla variabile
  - Nota: La valutazione delle espressioni deve essere fatta in fase di esecuzione perchè il valore delle variabili, in generale, non è noto in fase di compilazione

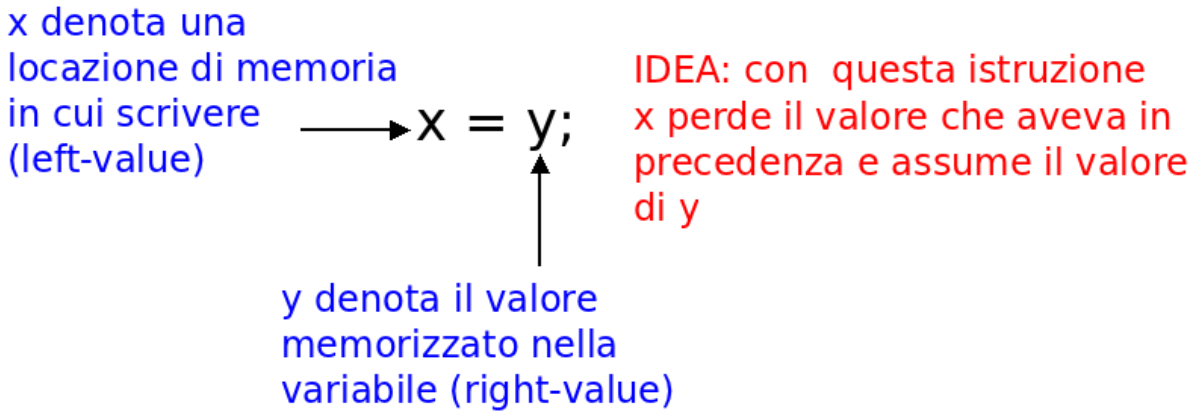


# Esempio

```
int a,b;

int main() {
    scanf(``%d'', &a);

    ...

    return 0;
}
```

- Suppiamo che al posto dei punti di sospensione vogliamo valutare l'espressione a
  - Qual è il tipo dell'espressione?
  - Qual è il valore dell'espressione?

# Come avviene l'assegnamento




Se il tipo dell'espressione è compatibile con il tipo della variabile allora in esecuzione accade quanto segue:

1. L'espressione viene valutata e se ne calcola il valore $\alpha$
2. $\alpha$ è il valore che viene memorizzato nella variabile

# Operatori binari tra espressioni intere

- +: operatore di addizione
- -: operatore di sottrazione
- *: operatore di moltiplicazione
- /: operarore per la divisione intera
- %: operatore per il calcolo del resto della divisione intera
- La precedenza degli operatori è eriditata dall'aritmetica
  - L'uso di parentesi tonde `(exp)` è consentito

# Operatori binari tra espressioni in virgola mobile

- `+`: operatore di addizione
- `-`: operatore di sottrazione
- `*`: operatore di moltiplicazione
- `/`: operarore per la divisione in virgola mobile
- La precedenza degli operatori è eriditata dall'aritmetica
  - L'uso di parentesi tonde `(exp)` è consentito
- In caso di combinazione tra un'espressione intera e una in virgola mobile, si ottiene un'espressione in virgola mobile
  - L'intero è automaticamente promosso a virgola mobile
  - Esempio: sia `a` una variabile intera. Allora `(a + 5.2)` è un'espressione di tipo `float`.

# Tipo Booleano

- Molti linguaggi di programmazione hanno un tipo chiamato `boolean` o `bool`
- Un valore booleano può essere `vero` (`true`) oppure `falso` (`false`)
- In C non è presente un tipo `boolean`, ma si usano gli interi
  - Un valore intero uguale a 0 codifica il valore booleano `false`
  - Un qualsiasi valore intero diverso da 0 codifica il valore booleano `true`

# Altri operatori su interi (ma visti come booleani)

- `exp1 && exp2`: assume valore diverso da 0 solo se entrambe le espressioni `exp1` ed `exp2` hanno valore diverso da 0 (AND)
- `exp1 || exp2`: assume valore uguale a 0 solo se entrambe le espressioni `exp1` ed `exp2` hanno valore uguale a 0 (OR)
- `!exp`: assume valore 0 se `exp` è diversa da 0, assume un valore diverso da 0 se `exp` ha valore 0 (NOT)

# Confronti

- I seguenti operatori confrontano valori numerici (float, int, double) e char (considerando l'ordine della tabella ASCII)
- `exp1 == exp2`: restituisce un valore int non-zero se il valore di `exp1` coincide con quello di `exp2`
- `exp1 != exp2`: è equivalente a `!(exp1 == exp2)`
- `exp1 >= exp2`: restituisce un valore int non-zero se il valore di `exp1` è maggiore oppure uguale a quello di `exp` (similmente per `>`, `<=`, `<`).

# Esercizi

Stabilire i valori delle variabili `c1, c2, c3` date le seguenti dichiarazioni `float a, b; int d; int e;` ed inizializzazioni `a=12.6; b=7.0;`

- `c1 = (b*2 > 18) || (a>=b);`
- `c2 = (a+b)<16 && (b<=10);`
- `c3 = ((d==6)&&(e!=0)) && ((d!= 6)||(e==0));`